Федеральное государственное автономное образовательное учреждение

высшего профессионального образования

**Национальный исследовательский университет «Высшая школа**

**экономики»**

**Московский институт электроники и математики Национального**

**исследовательского университета «Высшая школа экономики»**

Допущен к защите

Заведующий кафедрой ВСиС

\_\_\_\_\_\_\_\_\_\_\_ А.В. Вишнеков

«\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2013 г.

Орлов Иван Алексеевич

**МОДЕЛИРОВАНИЕ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ НА РАСПРЕДЕЛЕННОЙ СИСТЕМЕ ОБЪЕКТНО-АТРИБУТНОЙ АРХИТЕКТУРЫ**

Направление 23.01.00.68 - Информатика и вычислительная техника

Магистерская программа - Сети ЭВМ и телекоммуникации

Магистерская диссертация

|  |  |
| --- | --- |
| Научный руководитель | Салибекян С.М. |

*подпись*

|  |  |
| --- | --- |
| Рецензент | Панфилов П.Б. |

*подпись*

Москва 2013

Аннотация.

В диссератации описаны принципы имитационного моделирования разработанной в МИЭМ НИУ ВШЭ объектно-атрибутной (ОА) архитектуры вычислительных систем (ВС), реализующих принцип управления вычислениями с помощью потока данных (dataflow-парадигма), и результаты имитационного моделирования dataflow-ВС, полученные в рамках выполнения НИР по разработке и моделированию архитектуры суперкомпьютерной dataflow-ВС по Федеральной целевой программе «Исследования и разработки по приоритетным направлениям развития научно-технического комплекса России на 2007-2013 годы» при финансовой поддержке Минобрнауки России. Описание dataflow-прадигмы приводится во введении.

Для имитационного моделирования dataflow-ВС была расширена среда программирования и имитационного моделирования, основанная и работающая принципах ОА-архитектуры. С использованием данной среды было проведено имитационное моделирования ОА-суперкомпьютерной системы. Описание ОА-архитектуры и среды программирования и моделирования приводится в разделе №1.

Также, с целью оценки производительности и определения различных параметров вычислительной системы, в данной среде были разработаны тесты, решающие задачи разных классов. Принципы моделирования и разработка тестов описаны в разделах № 2 и № 3 соответственно.

Все тесты были запущены в данной среде, результаты тестирования и их анализ приведены в разделе № 4.

Содержание

1 Введение 4

[Dataflow-парадигма 4](#_Toc359187704)

[Анализ существующих решений в области dataflow-ВС 5](#_Toc359187705)

[Выводы по анализу программных решений dataflow 7](#_Toc359187706)

2 ОА-архитектура 19

2.1 [Описание программной реализации оа-архитектуры 20](#_Toc359187716)

[2.2 Устройства ядра ОА-архитектуры dataflow-ВС 24](#_Toc359187717)

3 Принцип моделирования 28

[3.1 Выбор и обоснование критериев оценки моделируемой системы 28](#_Toc359187718)

[3.2 Выбор и обоснование параметров моделируемой системы 28](#_Toc359187719)

[3.3 Моделирование систем на основе ОА-архитектуры 30](#_Toc359187720)

4 Разработка тестов 36

4.1 Тестирование архитектуры с помощтю задач расчет напряженности электростатического поля 36

[4.2 Бенчмарк GRAPH500 на основе алгоритмов решения теоретико-графовых задач 40](#_Toc359187723)

[4.3 Бенчмарк GREP – алгоритм сопоставления файловых строк шаблонам команды 46](#_Toc359187725)

5 Проведение тестирования 57

5.1 Тестовые прогоны на модели ОА-архитектуры dataflow-ВС задач расчета напряженности электростатического поля 57

5.2 Тестовые прогоны на модели ОА-архитектуры dataflow-ВС задач бенчмарка GRAPH500 на основе алгоритмов решения теоретико-графовых задач 64

5.3 Тестовые прогоны на модели ОА-архитектуры dataflow-ВС задач бенчмарка GREP – алгоритма сопоставления файловых строк шаблонам команды 71

5.4 Анализ результатов тестовых прогонов на модели ОА-архитектуры dataflow-ВС 75

Выводы 80

Список используемой литературы 81

1. Введение
   1. **Dataflow-парадигма**

В архитектурах вычислительных сетей на сегодняшний день преобладающую роль играют ВС, управляемые потоком команд – Control Flow. Такая парадигма вычислений ориентирована на последовательные вычисления. Она основывается на понятии программного счетчика, который хранит адрес ячейки памяти, из которой считывается выполняемая на текущий момент времени команда. Признаки control-flow-архитектуры видны также и на программном уровне: в настоящее время наиболее популярными языками программирования являются императивные языки (Pascal, С, C++, Java и др). Было предпринято множество попыток создания аппаратных систем, управляемых потоком данных, однако особых успехов они не принесли, т.к. ядром подобных систем являлся вычислитель фон-неймановской архитектуры.

Одним из перспективных способов оптимизации вычислительного процесса является применение dataflow-парадигмы, т.е. метода управления вычислительным процессом потоком данных. Данная парадигма известна уже более тридцати лет. Работы по аппаратной и программной реализаций этой концепции проводились различных исследовательских центрах. Например, в Массачусетском технологическом институте (процессор Tagget Token), лабораториях корпорации Tеxas Instruments (США), в Манчестерском университете (Англия). Но всё же наивысшим успехом данных проектов было создание экспериментальных прототипов.

В dataflow-системах очередность выполнения вычислительных задач зависит от готовности операндов. Как только необходимое число операндов приходит на исполнительное устройство, оно тут же начиняет выполнять свою вычислительную задачу. Стоит отметить, что очередность исполнения команд может коренным образом отличаться от очередности записи команд в программе. Налицо одно из главных преимуществ data-flow парадигмы: разные итерации одного цикла могут выполняться одновременно, на разных вычислительных узлах. Более того, (i+1)-я итерация цикла способна начаться исполняться быстрее, чем i-я, при условии наличия всех необходимых операндов. Система состоит из множества исполнительных устройств, каждое из которых может быть настроено «узко»: например, на выполнение вычислительных операций лишь одного типа.

С помощью данной концепции возможно преодолеть слабое место скалярных процессоров, которое заключается в потери времени на процедуры управления данными и спекулятивные вычисления.

* 1. **Анализ существующих решений в области dataflow-ВС**

В истории науки проводились попытки создания эффективных систем dataflow не только на аппаратном уровне, но и на уровне программном. В реализации на программном уровне были достигнуты некоторые положительные результаты: создание объектно-ориентированной парадигмы программирования (ООП), частично работающий по событийному принципу (в особенности язык Smalltalk); некоторые функциональные языки (например, Mozart, Sisal). Итак, перечислим некоторые решения, связанные с программной реализацией принципа dataflow.

**Сети Кана**

Считается, что предтечей всех dataflow-языков являются сети Кана (англ. Kahn) (Рисунок 1), в которых вычислительный процесс представляется в виде модулей (акторов) обменивающихся между собой сообщениями по линиям связи, установленными между ними. На линиях связи имеются очереди сообщений для того, чтобы в них помещались сообщения, которые ввиду занятости модуля-приёмника, не могут быть в данный момент приняты на обработку. Данная модель вычислений была предложена в 1974 году.



Рисунок 1 – Процессная сеть Кана

**U-язык**

Данный язык (Рисунок 2) используется для описания токенов, передаваемых по дугам потокового графа в машинах с памятью фреймов.

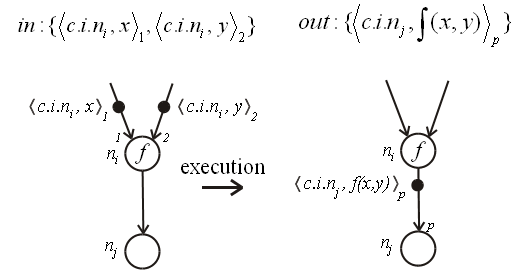


Рисунок 2 – U-язык

Каждый токен включает в себя два поля:

- активное имя;

- данные.

Активное имя, в свою очередь, включает в себя тег, который состоит из полей:

- адрес инструкции (n);

- поле контекста (c), уникально идентифицирующий контекст, в который входит инструкция;

- номер итерации (i), который идентифицирует активную итерацию цикла;

* в том случае, если узел, куда пересылаются данные, имеет несколько входных аргументов, то в тег включается номер порта назначения узла-адресата (p).

**LUCID (1976)**

Язык основан на концепции потока: даже константы, представляют собой поток (числовые константы – это поток состоящий из повторений данной числовой константы). Язык по сути функциональный. Потоки могут быть как бесконечные, так и конечные (конечным поток будет, если в качестве его хвоста использована специальное значение eod (end of data). Арифметические операции применяются к потокам поэлементно.

Одна из версий языка под названием Granular Lucid (GLU), разработанная в калифорнийской компании SRI International, предназначена для описания алгоритма крупнозерниского dataflow: dataflow-программа пишется на языке LUCID, а «зерна» - на языке Си. «Зёрна» оформляются в виде функций на языке Си, расположенных в отдельном файле.

**ID (IRVINE DATAFLOW) 1978**

Язык используется для программирования dataflow-систем Массачусетского технологического института (MIT) и системы Monsoon фирмы Motorola. Концепция данного языка высокого уровня была предложена Арвиндом и Гостлоу (англ. Arvind and Gostelow). Язык предназначен для программирования параллельных архитектур, в том числе, и динамического dataflow. Основные черты языка: однократное присвоение значений переменным, блочная структуризация программы, функциональная парадигма. Типы переменных не задаются напрямую, а определяются по значению, которое было в них помещено. Структуры данных и массивы не различаются между собой: элементы структуры могут адресоваться как с помощью индекса, так и с помощью строковых идентификаторов (например, «, t[‘‘height’’]»). Над структурами данных определены две операции select (выбор), предназначенный для того, чтобы получить значение из поля данных структуры, и append (добавить), необходимая, чтобы создать новую структуру путём копирования полей другой структуры данных или чтобы записать значение в поле структуры данных.

Id-программа состоит из набора свободных выражений и подвыражений, которые могут выполняться в любом порядке в зависимости от наличия данных (операндов) для их выполнения.

**VAL** **(A Value-oriented Algorithmic Language) 1979**

Язык был разработан а Массачусетском технологическом университете. VAL – это язык программирования высокого уровня, который не содержит машинозависимых элементов, что обеспечивает переносимость программы на различные аппаратные платформы. Язык предназначен для реализации статической потоковой архитектуры (следовательно, не реализуются рекурсивные вызовы). По синтаксису напоминает язык Pascal. Функции и выражения могут возвращать сразу несколько значений. У каждой переменной имеется специальное значение «Ошибка». Это нужно для того, чтобы сохранить необходимый порядок вычислений: если значение переменной еще не получено, то переменная содержит значение «Ошибка». Язык обеспечивает скрытое описание параллелизма. В данном языке переменные могут определяться только один раз (до определения в переменной хранится значение «пусто», а после присвоения значения, содержимое переменной меняться уже не может). Во время декларации массивов их размерность не указывается: размерность можно задать и определить с помощью специальных команд.

В VAL весьма ограничены возможности ввода/вывода и не поддерживается рекурсия из-за того, что программа транслируется напрямую в потоковый граф.

**SISAL (Streams and Iterations in a Single Assignment Language) 1983 г.**

Язык SISAL ориентирован на поддержку научных вычислений и представляет собой дальнейшее развитие языка VAL. Язык появился на свет благодаря сотрудничеству Ливерморской национальной лаборатории имени Лоренца, Университета штата Колорадо, Манчестерского университета (Великобритания) и Digital Equipment Corporation (DEC). Создатели языка ставили перед собой следующие задачи: 1. язык должен эффективно работать на машинах как последовательной, так и параллельной архитектур; 2. обеспечить создание независимого от языка и целевой архитектуры промежуточного представления программы в виде потокового графа; 3. разработать технику оптимизации параллельных программ; 4. внедрить функциональный стиль программирования для сложных научных расчётов; 5. обеспечить программирование микропроцессорных ВС с распределённой памятью.

Хотя SISAL и относится к разряду функциональных языков, однако ему присущи и элементы декларативной парадигмы программирования. Причём декларативная часть программы свободна от побочных эффектов: программа представляет собой набор последовательных императивных процедур, связанных между собой обменом данными. Оптимизирующий компилятор языка SISAL освобождает пользователя от работы по распараллеливанию программы, беря на себя всю нагрузку по планированию операций, передачу данных, синхронизацию вычислений, управлению памятью, что позволяет программисту сконцентрироваться именно на описании логики программы, не больше ни на чём.

Язык получил большую популярность в 90-е годы благодаря тому, что существовал компилятор, который мог переводить программу на этом языке в язык С, и потому программы, написанные на SISAL могли запускаться на любой машине, где был реализован С-компилятор. Еще одна причина популярности языка кроется в весьма удачной концепции модулей и интерфейсов, когда описание функций находится в интерфейсе, а алгоритм самих функций помещается в модуле.

**POST**

Язык дает обширные возможности по программированию:

- возможность влияния на стратегию вычислений уже в процессе выполнения программы;

- структуры данных, являющие одновременно синхронными и асинхронными;

- взаимодействие между вычислениями, чтобы прекратить вычисления по лишним ветвям алгоритмов.

Однако синтаксис программы чрезвычайно неудобен для понимания.

**ПИФАГОР 1995**

Этот функциональный язык разработан в Красноярском Государственном Техническом Университете (КГТУ).

Программа в ПИФАГОР представляет собой граф программы, описываемый с помощью специальных языковых конструкций. Данные передаются по дугам графа посредством токенов, переносящих как скалярные и векторные величины, так и функции преобразования данных.

Модель вычислительного процесса задается с помощью тройки

**M = ( G, P, S0 ),**

где **G** – ациклический граф, что определяет информационную структуру программы (ее информационный граф),

**P** - набор правил, определяющих динамику функционирования модели (механизм формирования разметки),

**S0** – начальная разметка.

Информационный граф G, в свою очередь, задаётся двойкой:

**G = ( V, A ),**

где **V** - множество вершин определяющих программо-формирующие операторы,

**A** - множество дуг, задающих пути передачи информации между ними.

Программно-формирующие операторы, являющиеся вершинами графа, осуществляют информационные преобразования данных, их структуризацию и размножение. Существуют следующие типы операторов:

- Интерпретации. Узел данного типа имеет два входа (по одному поступает аргумент, что требуется преобразовать с помощью функции, код которой поступает на второй вход; преобразованная величина поступает на единственный выход).

- Константный оператор. Данный узел выдает фишку с константой.

- Копирования данных. Копирует пришедшие на единственный вход данные на несколько выходов.

- Группировки в список: данные, пришедшие на несколько входов, группируются в один список.

- Группировки в параллельный список: пришедшие данные группируются в параллельный список (элементы параллельного списка рассматриваются оператором интерпретации как независимые аргументы).

- Группировки в список задержанных вычислений. (На рисунке данный оператор представлен в виде прямоугольника, выделяющего определенный подграф). С помощью этого оператора в графе программы выделяется отдельный подграф, выполнение которого активизируется не только по приходе всех необходимых для вычисления данных, но и при снятии задержки, т.е. разрушении границ подграфа.

Ввиду того, что программа ориентирована на непосредственное описание графа программы, синтаксис языка весьма отличается от общепринятого синтаксиса. Например, в языке не поддерживаются инфиксные бинарные операции, отсутствуют модульное построение программы, раздельная трансляция и интеграция стандартных библиотек, что не могло сказаться на популярности данного языка программирования. Отсутствие циклов в языке Пифагор компенсируется возможностью использования рекурсии. А сочетание механизма перегрузки функций и динамических типов, определяемых пользователем, обеспечивает написание эволюционно расширяемых параллельных программ. К тому же в языке реализован механизм перегрузки функций, что позволяет производить безболезненное масштабирование программы.

**Mozart**

В данном языке программирования, поддерживающим множество парадигм вычислений, реализуется и стратегия dataflow. Для этой цели существует команда Wait, которая используется для синхронизации по данным нескольких параллельных нитей вычислений. Нить, в коде которой встречается команда Wait(A), (A – переменная), дойдя до данной команды, будет ожидать, пока другая нить не запишет данные в переменную A. Как только данные в переменную будут записаны, нить продолжит свою работу. Принцип dataflow может использоваться в языке Mozart и более широко (Листинг 1): здесь в переменную S поступают значения — если переменная S пустая (не заполненная), то поток, реализующий данную программу останавливается до того момента, когда данные поступят (как очередная порция данных поступить нить вычислений возобновится, и снова остановится для ожидания следующией порции данных и так далее, пока не поступят все данные).

for Msg in S do

...

end

*Листинг 1. Считывание потока данных в языке Mozart*

**Smalltalk**

Язык Smalltalk основывается на следующих понятиях:

1. *Объект*. Вся программа состоит из совокупности объектов, обменивающимися сообщениями между собой. Объекты состоят из полей (данные) и методов (подпрограмм, обрабатывающих пошедшие сообщения). Каждому объекту соответствует свой протокол.

2. *Класс.* Однотипные объекты объединяются в классы, а каждый объект является экземпляром класса.

3. *Метод.* Подпрограмма, предназначенная для обработки сообщения, пришедшего от другого объекта. Каждому типу сообщения соответствует свой метод его обработки. Основная задача метода — при приходе сообщения соответствующим образом изменить состояние объекта и выслать ответ на пришедшее сообщение.

4. *Операторы.* Существует всего три оператора: посылка сообщения, выдача ответа, присваивание значения переменной. Синтаксис оператора посылки сообщения несколько напоминает синтаксис естественного языка (подлежащее - сказуемое - дополнение): объект-получатель ИмяСообщения, [объекты-атрибуты], например:

*5 factorial -* объекту 5 посылается сообщение с именем "factorial";

*а – 3 -* объекту "а" посылается сообщение с именем "-" и аргументом 3;

*pen move:east by:10* - объекту pen посылается сообщение "*move:eastby:10"* , где "*move"* и *"by" —* селекторы ключевых слов, по которым метод идентифицирует пришедшие с сообщением аргументы, а "*east"* и *"10" —* сами аргументы (":" - обозначение того, что после селектора идет указание параметра).

Хотя Smalltalk очень близок по духу к концепции dataflow, однако он до конца, все же, не уходит от командной парадигмы: по сути сообщение — это завуалированное описание одной фон-неймановской команды. Можно сказать, что программная среда Smalltalk создает среду из виртуальных объектов, которые обмениваются между собой командами.

**DCF**

Язык DCF – это расширение языка C для работы в режиме dataflow. Модель, реализованная в DCF, относится к классу data-control flow (система, сочетающая в себе модель управления вычислениями с помощью потоков команд и данных). DCF зиждется на двух ключевых понятиях: нить и токен. Нить является классическим легковесным процессом (цепочка последовательно выполняемых команд). Нить начинает выполняться, как только для программы этой нити приходят все необходимые данные. Функция, реализуемая в виде отдельной нити вычислений, описывается с помощью ключевого слова «thread» вместо типа возвращаемого значения: thread Func( int N, float Var ) { … }. Выполнение программы начинается с функции-потока с именем main. Функции-нити могут порождать другие нити; выполнение программы считается завершённым, когда завершаются все вычислительные нити.

Данные же передаются с помощью токенов, которые состоят из тега, идентифицирующего получателя данных; самих данных типов int, float, char языка Си, другую описывающую информацию о передаваемых данных. Тег состоит из двух полей: имя функции-получателя данных и «цвет», задающую номер итерации цикла. С помощью токенов происходит запуск и возобновление работы функций. Токены «обитают» в неком «пространстве» токенов, где они хранятся и где происходит их сбор и объединение в группы: каждая группа — это набор токенов, необходимых для запуска определённой нити вычислений (полностью скомплектованная группа содержит токены, хранящие данные для всех аргументов активируемой функции-потока).

**Модель акторов**

Модель исходит из того, что вся программа делится на несколько независимых программ (акторов), обменивающихся между собой сообщениями: если актору приходят данные необходимые для выполнения одного из заложенных в него привил, он производит вычисление и выдает сообщение с результатами другим акторам по специальным линиям связи (реальным или виртуальным). Актор может иметь внутреннее состояние и менять его в соответствии с заложенным в него алгоритмом. Во избежание потери информации, организуются очереди сообщений: если актор занят обработкой данных, то пришедшее сообщение помещается в очередь. Модель весьма хорошо подходит для реализации на распределенных ВС, т.к. каждый актор может существовать в изолированном адресном пространстве (связь осуществляется только через очереди сообщений). Термин "актор" был впервые предложен Карлом Хьюиттом (Carl Hewit) в 1970-х годах, который пытался с его основе разработать концепцию «мыслящих исполнительных устройств» (autonomous reasoning agents). Идея акторов применительно к модели параллельных вычислений была затем развита в научных работах Агха.

Акторная модель состоит из двух составляющих:

* + 1. Описание акторов (в описание входят интерфейс актора (описание входов и выходов актора), параметры (они обычно задаются перед началом вычислений и во время вычислительного процесса не меняются), алгоритм получения выходных результатов);
    2. Описание соединений акторов, по которым они обмениваются сообщениями (задаётся с помощью XML-файла, специального графического редактора или программы, использующей специальных аппаратно-программный интерфейс (например, SystemC)).

Приведём описание актора на языке Caltrop (листинг 2):

actor AddAndMultiply2[T] (T k) T A, T B → T C :

action [a], [b] → [k ∗ (a + b)] : endaction

action [a], [] → [k ∗ a] : endaction

action [], [b] → [k ∗ b] : endaction

endactor

*Листинг 2. Описание актора на языке Caltrop*

Здесь в качестве входных значений выступают A и B, а в качестве выходного — C, в качестве параметра актора выступает k (T – обозначение типа переменной); и описаны три действия (action), которые активизируются при:

* + 1. приходе двух операндов;
    2. приходе только операнда А;
    3. приходе только операнда В.

Данный подход уже давно используется в системах Simulink от The Mathworks, LabVIEW от National Instruments, ML-Designer, ПИФАГОР, Erlang, Caltrop и т.д.

**Графическое программирование**

Программы, выполненные в данной парадигме, представляют алгоритм не в виде текста, а в графическом виде: элементарные операторы и сложные функции иллюстрируются с помощью определённых графических объектов, а передача операндов (скалярных величин или информационных конструкций) от одной операции или функции к другой — графическими линиями связи (Link). Во время выполнения программы по link-ам будут передаваться токены с данными или ссылками на информационные конструкции. Можно сказать, такая графическая модель как раз и служит отражением графа потока данных. В качестве представителей данной парадигмы следует упомянуть системы программирования LabView (Рисунок 3) и MLDesigner (Рисунок 4). Первая система применяется для решения любых вычислительных задач, вторая — в основном для моделирования различных систем. Данный подход весьма похож на модель акторов (графические элементы являются акторами, которые по виртуальным линиям связи обмениваются различными сообщениями). Родоначальником такой парадигмы программирования можно считать Джека Денниса, который предложил описывать алгоритмы dataflow-машин с помощью изображения потокового графа.

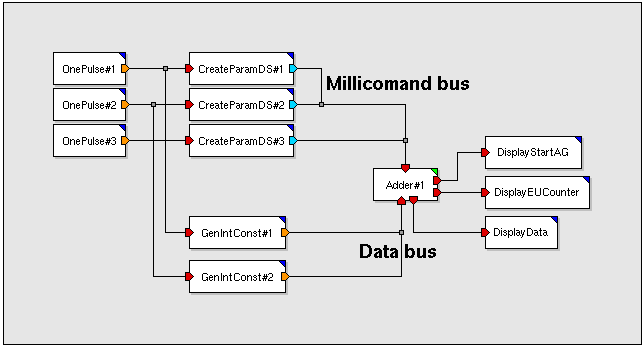
 

Рисунок 3 – Среда LabView Рисунок 4 – Среда моделирования MLDesigner

* 1. **Выводы по анализу программных решений dataflow**

Как было показано выше, в основном для реализации систем dataflow применяется функциональная парадигма программирования. Это объясняется тем, что данная парадигма не задаёт чёткой «траектории» вычислений (т.е. последовательности операций), как в классической процедурной (императивной) парадигме вычислений. Именно отсутствие программного счётчика и позволяет произвести полноценное распараллеливание вычислений, и управление вычислительным процессом с помощью потока команд. Однако функциональная парадигма в среде программистов, к сожалению, не получила должной популярности. Чрезвычайно популярная объектно-ориентированная же парадигма больше тяготит к процедурному стилю программирования и не обеспечивает полноценную работу в режиме dataflow. Единственным объектно-ориентированным языком, полностью соответствующим парадигме управления вычислительным процессом с помощью потока данных, является Smaltalk, который, к сожалению, не очень-то любим в среде программистов.

Многие языки, что разрабатывались под конкретные dataflow-машины (например, VAL), со временем были забыты. Универсальные же языки (например, SISAL), которые поддерживали кроссплатформенность развиваются и до сих пор.

Весьма выгодным оказывается направление по созданию гибридных покового-командных языков, таких как DCF: они позволяют программисту работать в привычной процедурной (императивной) парадигме, по возможности добавляя элементы dataflow-стиля.

Акторная модель весьма удобна для реализации на гетерогенных ВС, однако она имеет очень существенный недостаток: программа делится на множество маленьких кусков, которые не очень-то легко воспринимаются как единое целое.

В заключение следует отметить, что до сих пор не создано dataflow-языка программирования, который был бы безоговорочно принят в программистской среде — сейчас наиболее популярна процедурная (императивная) и объектно-ориентированная парадигмы, которые соответствуют концепции control flow.

1. ОА-архитектура

В данной работе предлагается использование объектно-атрибутной архитектуры ВС (или ОА-архитектуры). В отличие от классической ВС, ОА-архитектура работает по иным принципам, т.к. не имеет фон-неймановской архитектуры вычислительного ядра. Более того, ОА-архитектура может быть применима как к аппаратной, так и к программной частям ВС. Для аппаратной части приведем описание атрибутной (А-архитектурой), а для программной – описание ОА-архитектуры.

В атрибутной архитектуре вычислительная аппаратура представляется в виде набора функциональных устройств (ФУ). Взаимодействуют эти устройства с помощью информационных пар (ИП) – совокупности данных и характеризующего их атрибута. По сути, ИП представляет собой простейший токен. Такая организация вычислительной сети позволяет отойти от использования стандартных фон-неймановских команд, и, следовательно, обойти фон-неймановскую архитектуру в целом. Это позволяет намного повысить производительность вычислительной системы при выполнении параллельных вычислений.

Для программной реализации можно модернизировать А-архитектуру:

1. Несколько информационных пар более оптимально группировать в единую структуру данных – капсулу. С помощью этой группировки можно описывать объекты в рамках одной конструкции данных.
2. Функциональные устройства заменяются на виртуальные функциональные устройства (ВФУ). Для реализации удобнее разработать универсальный интерфейс ВФУ (например, в виде подпрограммы или класса).

Данные изменения способствуют применению программной реализации в качестве среды моделирования: количество виртуальных функциональных устройств может быть ограниченно только ресурсами вычислительной машины, на которой проводится моделирование.

Однако, для проведения моделирования на ОА-архитектуре требуется отличный от классического подход к программированию ВС. Это связано с совершенно иным способом управления вычислительным процессом, которым в данной архитектуре управляют сами ВФУ.

# 2.1 Описание программной реализации ОА-архитектуры

В реалиазации милликомандного типа управления вычислительной системой основную роль играет функциональное устройство «Автомат». Это устройство отвечает за выдачу команд на шину данных, через которую милликоманды доставляются необходимым виртуальным устройствам системы. С помощью данного ВФУ могут быть реализованы различного рода алгоритмические операции: циклы, условный переход, безусловный переход и др.



Рисунок 5 - Простейшая ОА-система

Также в системе присутствуют специальные виртуальные устройства для реализации распределенной работы вычислительной системы, такие как: виртуальное устройство «Маршрутизатор», «Шлюз», «Шина».



Рисунок 6 - Структура распределенной ОА-системы

При работе распределенной вычислительной системы, милликоманды, которые поступают на шину могут быть доставлены (в зависимости от адресации) как на некоторое виртуальное устройство, непосредственно связанное с шиной, так и на другие устройства, которые могут находиться на удаленных вычислительных узлах. Для этого Шина содержит пару внутренних регистров: LowMkRange и HighMkRange, которые хранят верхний и нижний диапазоны (соответственно) вычислительного сегмента. Каждая Шина, находящаятся в системе, хранит собственные значения этих регистров. При попадании милликоманды в диапазон между значениями этих регистров, вычисляется адрес ФУ-«адресата», для которого предназначена пересылаемая милликоманда. Если индеск команды выходит из диапазона, то Шина передает данную команду Маршрутизатору, который уже отвечает за доставку милликоманды.

В реализации маршрутизации данных в объектно-атрибутной системе используется разделение адресного пространства между вычислительными узлами. Для этого каждому сегменту такого разбиения выделяется диапазон адресов, который хранится в специальной таблице маршрутизатора.



Рисунок 7 - Структура маршрутизатора ОА-системы

После того, как функциональное устройство Маршрутизатор получает милликоманды, оно выполняет проверку полученной команды на вхождение в диапазон какого-либо из имеющихся каналов связи. Если вхождение выполняется, то милликоманда пересылается по данному каналу.

Виртуальное функциональное устройство Шлюз отвечает за обмен данными между распределенными вычислительным узлами системы. В соответствии с реализованным протоколом (например, TCP, UDP) шлюз передает данные по каналу связи.

Все вычисления проводятся непосредственно на функциональных устройствах.

Управление вычислительным процессом и сами вычисления в ОА-архитектуре осуществляется функциональными устройствами (ФУ). ФУ можно разделить на два класса: исполнительные (выполняют вычисления) и служебные (осуществляют управление вычислительными процессом). Служебные ФУ формируют ядро ОА-системы, управляющее работой вычислительного узла.

**2.2. Устройства ядра ОА-архитектуры dataflow-ВС**

Ядром вычислительной dataflow-системы будем называть совокупность оборудования, которое осуществляет сбор данных для формирования исполняемого пакета. В классических системах ядром является устройство на основе ассоциативной памяти, которое осуществляет сбор и поиск токенов, формирующих исполняемый пакет. Под контролем ядра находятся несколько исполнительных устройств – после того, как ядро набирает комплект данных для выполнения операции, оно формирует исполняемый пакет, определяет исполнительное устройство, которые будет его обрабатывать, и посылает пакет на обработку исполнительному устройству.

Главную роль в организации ядра объектно-атрибутной системе играет совокупность функциональных устройств под управлением ФУ Менеджера. В его задачи входит определение функционального устройства, для которого предназначен вновь пришедший токен, иными словами, информационная пара, а также передача данной ИП и прием ИП с других ядер. Полный пакет данных собирается во внутренних регистрах функционального устройства.

На рисунке 8 приведена структурная схема объектно-атрибутной dataflow-вычислительной системы, которая управляется Менеджером функциональных устройств. На схеме видно, что Менеджер принимает данные с различных устройств, и, на основе содержащейся в них информации, выполняет планирование ресурсов системы. Данное функциональное устройство состоит из двух частей: внутреннего контекста и реализации логики работы. Реализовать требуемый функционал ФУ-менеджера можно при помощи нескольких исполнительных устройств: либо универсального ядра процессорного ядра, либо при помощи специального устройства, которое могло бы реализовывать логику работы нескольких функциональных устройств различного типа. Очевидно, чтоиспользуя специализированные устройства можно добиться максимальной производительности системы, однако, такой подход увеличивает стоимость аппаратных средств для реализации.

**

Рисунок 8 - Структура ядра объектно-атрибутной системы

Планирование вычислений в функциональном устройстве Менеджера, происходит следующим образом:

1. Диспетчер хранит во внутренних регистрах очередь запросов на предоставление ресурсов тому или иному устройству. Если для функционального устройство, от которого пришел очередной запрос, не предоставлено исполнительное устройство, а другие ИУ заняты, то данный запрос помещается в очередь.
2. Когда одно из исполнительных устройств освобождается, т.е. некоторое функциональное (виртуальное) устройство завершает свою работу, то на данное исполнительное устройство загружается контекст виртуального устройства, находящегося в голове очереди ожидания ресурсов. Устройство, которому были предоставлены устройства будем называть активным.
3. В состав Диспетчера также входят специальные очереди милликоманд для ФУ. Некоторая милликоманда будет попадать в данную очередь в тех случаях, когда она была адресована виртуальному устройству, которому не были предоставлены аппаратные средства, и которое, в свою очередь, не находится в состоянии ожидания ресурсов (в таких случаях Диспетчер ставит функциональное устройство в очередь запросов, а пришедшую милликоманду помещает в очередь запросов для данного ФУ), или функциональному устройству, которое находится в очереди ожидания ресурсов, или устройству, для которого были предоставлены аппаратные ресурсы.
4. В случаях, когда очередь запросов на предоставление ресурсов покидает активное устройство, милликоманды, предназаченные для этого устройства переходят на выполнения на данном устройстве.

Т.к. функциональное устройство самостоятельно уведомляет диспетчер устройств о заверши совей работы, можно утверждать, что в архитектуре используется невытесняющий параллелизм. Однако, предусмотрены те аварийные случаи, когда диспетчер сомостоятельно отнимает ресурсы у активного устройства.

Вычислительная среда, разработанная для программирования и моделирования, работает по принципу ОА-архитектуры. Данная среда обладала следующими возможностями:

* Запуск и отладка программ, написанных на параллельном ОА-языке
* Создание виртуальных ФУ из общего набора реализованных ФУ
* Сохранение и восстановление ОА-образа ВС

Однако, для проведения имитационного моделирования на данной системе этих функциональных возможностей было недостаточно. Поэтому среда была доработана для решения следующих задач:

* Планирование вычислений виртуальных функциональных устройств между исполнительными устройствами
* Сбор сведений о моделируемом процессе при запуске параллельного вычислительного процесса для анализа производительности ВС

Для реализации перечисленных возможностей были разработаны дополнительные ФУ, обеспечивающие их выполнение.

1. Принцип моделирования

# 3.1 Выбор и обоснование критериев оценки моделируемой системы

Основными критериями оценки созданной вычислительной системы с управлением потоком данных по результатам прогона разнообразных тестовых задач на модели системы являются следующие:

- Время выполнения теста.

- Коэффициент параллелизма в текущий момент модельного времени Kp.

- Средний коэффициент параллелизма () – среднее число ИУ, работающих во время тестового прогона.

- Дисперсия коэффициента параллелизма D().

- Аппаратные затраты (число ИУ в системе) (NFU).

- Коэффициент использования аппаратуры KHW=

Время выполнения теста показывает быстродействие системы на определенном классе задач при заданной архитектуре моделируемой ВС.

Коэффициент параллелизма в текущий момент модельного времени Kp показывает, сколько в определенный момент времени было задействовано ИУ в вычислительном процессе. С помощью этого показателя можно отследить динамику ВП.

Средний коэффициент параллелизма показывает среднее число задействованных ФУ в единицу времени. Коэффициент вычисляется по формуле:

,

где Kpi – число одновременно работающих ФУ в i-м временном отрезке; – длина i-го временного отрезка.

Дисперсия коэффициента параллелизма D() показывает неравномерность загрузки аппаратуры. Если загрузка неравномерна, то оборудование в определенные временные моменты ВС может оказаться перегруженным, а в другие – недогруженым. При выборе архитектуры и подборе параметров ВС для решения конкретной вычислительной задачи, следует стремиться к тому, чтобы дисперсия стремилась к нулю, т.к. большая дисперсия приводит к увеличению времени выполнения тестового прогона.

Коэффициент использования аппаратуры коррелирует с показателем стоимость/производительность, т.к. объем аппаратуры пропорционален стоимости ВС. Следовательно данный показатель следует повышать: в частности, для увеличения коэффициента использования аппаратуры в случае адаптации широкого класса физических задач, решаемых с помощью сеточных методов, используется запуск нескольких волн (итераций) по вычислительной сетке.

# 3.2 Выбор и обоснование параметров моделируемой системы

Оценка моделируемой ВС осуществляется на основе анализа функционирования ВС на тестовых задачах по следующему набору параметров:

- общее число ИУ в ВС;

- распределение ИУ по ядрам;

- топология ВС (взаимосвязи ядер между собой);

- время передачи данных по каналу связи;

- время работы ФУ;

- время планирования вычислений (время, за которое Планировщик загружает ФУ в ИУ для выполнения милликоманды).

Общее число ИУ определяет пиковую производительность ВС.

Распределение ИУ по ядрам также может оказывать влияние на производительность ВС, т.к. влияет на распределение информационных потоков в ВС.

Топология во многом определяет организацию потока данных в ВС, а передача информации сопровождается задержками, которые негативным образом сказываются на производительности ВС.

Время передачи по каналу связи может влиять на производительность ВС и, поэтому, является существенным фактором при моделировании.

Время работы ФУ во многом определяет критерии ВС. Время работы ФУ может зависеть от нескольких факторов, таких как тип ФУ и тип ИУ, на котором реализуется логика работы ФУ, а также объем и характер данных, передаваемых для ФУ.

Объем ОП, необходимый ФУ для вычислений, зависит от логики работы ФУ и от объема и характера поступающих к нему данных. Параметр важен, т.к. по нему оценивается максимальный объем памяти, необходимой для прохождения заданных тестов, т.е. объем ОП необходимой для реализации ВС.

Время планирования также может существенным образом сказываться на критериях оценки ВС, так как балансировщик (планировщик) является устройством (или программой), для осуществления функций которого требуется время.

**3.3 Моделирование систем на основе ОА-архитектуры**

Моделирование различных вычислительных систем можно разделить на два главенствующих класса: матечатическое моделировании и имитационное. Математическое моделирование предусматривает построение модели вычислительной системы в виде и математических формул и формальных конструкций, описывающих поведение моделируемой системы. Имитационное моделирование образуют модели-имитаторы, цель которох состоит в описании «как они есть». Имитационную модель можно выполнять во времени. Используя такие модели, можно проводить ряд испытаний системы (событийность испытаний можно проводить случайным образом), собирать статистику различных параметров моделируемой системы и по ней определять поведение и характеристики системы в целом.

В свою очередь, имитационное моделирование делится на два класса: 1) ориентированное на события (с переменным квантом времени), когда состояние модели меняется во время какого-либо события; и 2) моделирование с постоянным временем, когда модельное время делится на временные такты одинаковой длительности.

Оба вида моделирования (формальное-математическое и имитационное) дополняют друг друга таким образом, что результаты математического моделирования могут использоваться для проверки качества разработанной имитационной модели (в идеале результаты этих двух видов моделирования должны быть максимально близки друг к другу).

**Принципы математического моделирования dataflow-ВС**

Для моделирования параллельных вычислительных систем удобно использовать сетевые модели, а в частности, модель сетей Петри. Однако сети Петри не могут полностью описать параллельные вычислительные системы и dataflow-системы. Карл Хьют (Массачусетский технологический университет, профессор), отмечает некоторые недостатки сетей Петри:

* сети Петри моделируют управление потоком, но не сам поток данных;
* сложность в описании одновремененных событий, которые происходят во время моделирования вычислительного процесса;
* Переход в сетях имеет весьма сомнительную физическую интерпретацию

В связи с перечисленными недостатками, актуальной была разработка нового формализованного аппарата описания вычислительных систем, работающих по принципам dataflow. В результате, Салибекяном С.М, и Панфиловым П.Б., был предложен новый формализм модели атрибутной сети [1], который приведен ниже:

А-сеть представляет собой восьмерку:

*A={I,C,O,EC,CO,OI,IM,OM},*

где I – множество входных узлов;

C – множество вычислительных узлов;

O – множество выходных узлов;

IC: I → C – множество дуг из входных вершин в вычислительные узлы;

CO: C →O – множество дуг из вычислительных узлов в выходные узлы;

OI: O → I – множество дуг, соединяющих выходные вершины со входными;

IM – вектор маркировок входных узлов;

OM – вектор маркировок выходных узлов.

В теоретико-графовой интерпретации А-сеть – это трёхдольный граф, где одна доля (множество C) – это вычислительные вершины (на них производится исполнение команды – в реальной ВС это – исполнительные устройства (ИУ)), вторая (множество I) – входные данные (в реальной систем это – внутренние регистры для хранения промежуточных данных в исполнительных устройствах), а третья (множество O) – выходные данные для вычислительных вершин (это – внутренние регистры, в которые исполнительные устройства помещают результат вычислений) (рис. 9).

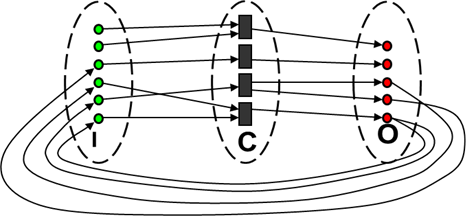
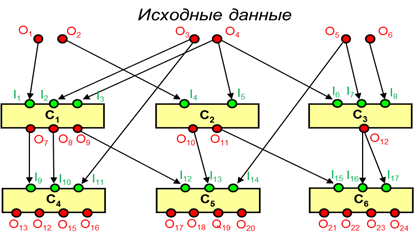


Рис. 9 – 3х-дольный граф атрибутной сети

На рис. 10 атрибутная сеть представлена в другом (ярусном) виде. Здесь вычислительная вершина обозначена прямоугольником, входные и выходные узлы обозначены кружками зеленого и красного цвета соответственно. Для входных и выходных вершин вводится маркировка, имеющая только два значения – «пусто» и «заполнено». «Пусто» означает, что во входную вершину еще не поступили данные, «заполнено» - поступили. Маркировка задается с помощью векторов маркировки входных (IM) и выходных (OM) вершин. Первый ярус входных вершин на рис. 10 обозначает входные данные для вычислительного процесса, последний ярус выходных вершин – выходные данные, полученные на выходе из ВС.

**

*Рис. 10 – Атрибутная сеть*

**Принципы имитационного моделирования систем на основе ОА-архитектуры**

Моделирование dataflow-вычислительной системы объектно-атрибутной архитектуры относится к классу событийно-ориентированного имитационного моделирования.

Применение объектно-атрибутной архитектуры позволяет производить программирование и отладку программных комплексов без подключения реальных аппаратных средств. Благодаря такой мобильности, объектно-атрибутный образ может быть запущен на любой вычислительной системе, (от мобильного устройства, до домашнего персонального компьютера.

Применяемый механизм информационных пар легко заменяет источники поступаемых входных данных в вычислительной системе. Набор входных информационных пар может быть определен программистом до запуска при моделировании. Реакцию системы можно отследить по данным, которыми обмениваются функциональные устройства, передавая их через шину данных. Также, для моделирования сложных вычислительных систем можно задавать время поступления определенной информационной пары в систему.

Во временн*о*м моделировании вводятся параметры задержки информационных пар. Для этого, каждая ИП дополняется определенной временной меткой, которая хранит модельное время передачи данной ИП. При реализации модели на аппаратном уровне, временные метки не используются.

Использование ОА-образа при моделировании системы демонстрирует все преимущества кроссплатформенности и переносимости ОА-вычислительной системы: после автономной отладки, ОА-образ может быть загружен в реальную вычислительную систему. Таким образом, автономное моделирование снижает временные и ресурсные затраты, которые тратятся при имитационном моделировании вычислительной системы.

1. Разработка тестов

В предыдущем разделе был приведен необходимый для получения набор выходных характеристик моделируемой вычислительной системы, требуемый от тестирования. Для получения наиболее объективных значений данных характеристик выгодно использовать тесты, построенные на задачах разных классов. По анализу популярных бенчмарков параллельных вычислительных систем можно выделить основные классы задач тестирования:

* Задачи, построенные на использовании сеточных методов
* Задачи, построенные на операциях с графами большой размерности
* Задачи, построенные на анализе и обработке больших текстовых данных

Для класса задач с использованием сеточных методов была выбрана задача расчета напряженности электростатического двумерного поля.

Для класса задач с использованием операций над графами был выбран популярный бенчмарк компании Graph500, который используется для построения рейтингов суперкомпьютеров.

Для класса задач с использованием обработки больших текстовых данных был выбран тест, основанный на поиске по тексту в соответствии с регулярным выражением.

**4.1 Тестирование архитектуры с помощью задач расчета напряженности элетростатического поля**

Данный тест можно разбить на следующие подзадачи:

1. Создание матрицы вычислительных узлов, каждый из которых моделирует некоторую точку на плоскости
2. Помещение (задание) некоторой точке определенной величины заряда. В данном случае моделируется помещение заряженной частицы в некоторую точку плоскости.
3. Расчет вектора напряженности в каждом узле матрицы (т.е. в каждой заданной точке плоскости). Расчет проводится с помощью нескольких диагональных «проходов» по матрице.



*Рисунок 12. Передача данных о зарядах на матрице функциональных устройств.*

После того, как некоторой точке матрицы был задан электрический заряд, ФУ, отвечающее за данную точку, запускает «волну просчета». Оно посылает своим четырем соседям данные о новом заданном заряде (величина заряда и его координаты). Другие ФУ, получая на вход данные о заряде, рассчитывают результирующий вектор напряженности и записывают его в память. По умолчанию, каждое ФУ хранит нулевой вектор. После расчета ФУ отправляет информацию некоторым своим соседям. Выбор направления для отправки зависит от координат текущей точки и координат заряженной частицы. Можно выделить 8 различных вариантов выбора направлений отправки:

1. Если ФУ находится в верхней левой части плоскости, то оно пересылает данные «соседу» слева
2. Если ФУ находится в верхней правой части плоскости, то оно пересылает данные «соседу» сверху
3. Если ФУ находится в нижней правой части плоскости, то оно пересылает данные «соседу» справа
4. Если ФУ находится в нижней левой части плоскости, то оно пересылает данные «соседу» вниз
5. Если ФУ находится на горизонтальной оси слева от ФУ, то оно отправляет данные «соседу» слева и вниз
6. Если ФУ находится на горизонтальной оси справа от ФУ, то оно отправляет данные «соседу» справа и вверх
7. Если ФУ находится на вертикальной оси сверху от ФУ, то оно отправляет данные «соседу» слева и вверх
8. Если ФУ находится на вертикальной оси снизу от ФУ, то оно отправляет данные «соседу» справа и вниз

Схема выбора направлений приведена на рисунке 13.



*Рисунок 13. Схема выбора направлений для отправки данных о новом заряде*

Такой алгоритм выбора направления для отправки данных исключает повторную доставку данных об одном и том же заряде одному ФУ. Таким образом, каждое ФУ получит число данных, соответствующее числу заданных зарядов на плоскости.

Каждый узел (ФУ) в своем сообщении посылает данные о векторе напряженности, соответствующему его точке на плоскости. Получив входные данные, узел вычисляет результирующий вектор напряженности. По умолчанию, в каждой точке записан нулевой вектор.

http://www.physics.ru/courses/op25part2/content/javagifs/63230164550370-15.gif

где

радиус-вектор предыдущего узла

Для реализации теста была написана следующая миллипрограмма:

NewFU={Mnemo="Eventser" FUType=FUEventser}

NewFU={Mnemo="Scheduler" FUType=FUScheduler}

NewFU={Mnemo="ElectricField" FUType=FUElectricFieldManager}

Eventser.ContextPopMk=Scheduler.EventserContextSet

Scheduler.ContextPopMk=ElectricField.SchedulerContextSet

ElectricField.NSet=100

ElectricField.MSet=100

ElectricField.EpsSet=1.5

ElectricField.NetCreat

\\ Расстановка точек с зарядом на плоскости

ElectricField.XSet=10

ElectricField.YSet=10

ElectricField.PotensialSet=5

ElectricField.XSet=99

ElectricField.YSet=36

ElectricField.PotensialSet=-5.1

ElectricField.XSet=3

ElectricField.YSet=61

ElectricField.PotensialSet=3.6

ElectricField.PotensialMatrPopMk=Console.LnOut

*Листинг 3. Реализация теста расчета электростатического поля*

**4.2 Бенчмарк GRAPH500 на основе алгоритмов решения теоретико-графовых задач**

Данный тест моделирует работу интернет-ресурсов социальных сетей. Социальная сеть представляется в виде графа, где узлы обозначают людей, а ребра – связи (знакомства) между ними. Таким образом строится тестовая задача на основе графовой модели. Первая часть задачи – это создание графа, т.е. синтез ребер графа, который осуществляется на основе деления матрицы смежности графа на четыре части (рис. 69).

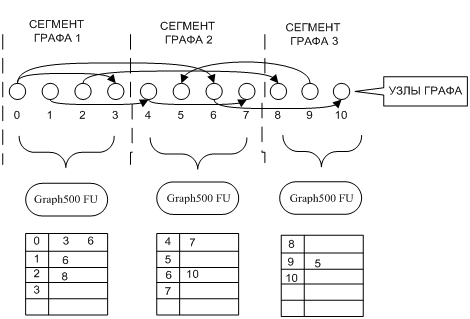


Рисунок 14 - Функциональная схема теста Graph500

За проведение первой фазы теста ответственно ВФУ Graph500Manager. Размерность матрицы смежности задается параметром Scale. Число вершин анализируемого графа N= 2Scale. В графе для теста синтезируется M ребер (обычно M=16\*N).

Для моделирования данной задачи для реализации на ОА-архитектуре dataflow-ВС были использованы следующие значения постоянных параметров:

* Число вершин графа: 210 = 1024
* Число ребер графа: 16\*210
* Число ФУ для реализации теста: 27 = 128
* Число итераций теста: 64

**ФУ для реализации приложения бенчмарка GRAPH500**

В и собраны наборы милликоманд ФУ, предназначенных для прогона тестовых приложений бенчмарка GRAPH500 на модели суперкомпьютерной системы ОА-архитектуры.

**Менеджер по управлению ФУ на тесте GRAPH500**

|  |  |  |
| --- | --- | --- |
| **Индекс милликоманды** | **Мнемоника милликоманды** | **Описание** |
| 0 | Reset | Сброс ФУ |
| 1 | ScaleSet | Установить scale (корень от числа вершин) |
| 5 | EdgeSet | Установить число ребер в графе |
| 10 | EdgeFactorSet | Установить число ребер с помощью коэффицента от числа вершин графа |
| 20 | CollectorSet | Установить ссылку на сборщика результатов |
| 25 | FUFieldRangeSet | Установить размерность поля для анализа графа (число ФУ на одной грани вычислительного поля) |
| 100 | Run | Начать тест (генерацию FU для анализа графа)  На входе – количество шагов моделирования  (по умолчанию прогоняется 64 итерации) |

*Таблица 1 - Милликоманды устройства «решателя графовых задач»*

**Сборщик результатов работы по тесту Graph500 (Graph500Collector)**

|  |  |
| --- | --- |
| **Мнемоника милликоманды** | **Описание** |
| Reset | Сброс ФУ |
| GraphGenSet | Устновить ссылку на генератор графа |
| PointSet | Принять номер вершины с итераций |
| CountPopMk | Выдать милликоманду со значением счетчика вершин |
| TraceLongPopMk | Выдать милликоманду со средней длиной пути в графе |
| IterationSet | Установить номер текущей итерации |
| TraceLongIterationPopMk | Выдать длину пути для итерации |
| TraceLongIterationPopMkList | Выдать длину пути для итерации |

*Таблица 2 - Милликоманды устройства сбора данных для решателя графовых задач.*

Программирование тестовой задачи бенчмарка GRAPH500 для суперкомпьютерной системы ОА-архитектуры осуществляется с помощью следующей миллипрограммы на ПЯ, особенностью которой является то, что она позволяет варьировать во время моделирования число ИУ в моделируемой системе.

NewFU={Mnemo="Scheduler" FUType=FUScheduler}

Scheduler.ContextPopMk=MainBus.SchedulerContextSet \\ Настройка ссылки на балансир (планировщик)

NewFU={Mnemo="Eventser" FUType=FUEventser}

NewFU={Mnemo="GraphGen" FUType=FUGraph500Manager}

NewFU={Mnemo="GraphCollector" FUType=FUGraph500Collector}

NewFU={Mnemo="Chart" FUType=FUChart}

NewFU={Mnemo="Plot" FUType=FUPlot}

\\ Настройка рисования графика

Plot.SeriaCreat

Chart.ParentSet

Chart.Caption="Parallel factor"

Chart.Focus

Plot.SeriaCreat

Chart.ChartPopMk=Plot.ParentSet

Plot.ColorSet=clRed

Plot.PenWidthSet=3

Chart.TitleYSet="Parallel factor"

Chart.TitleXSet="Model time"

Chart.TitleSet="Graph500 parallel factor"

Scheduler.SheduleTimeSet=0

\\ Подпрограмма вывода параметра в окно графика

Eventser.OutProgSet={Eventser.CurrentTimePopMk=Plot.XSet Eventser.CurrentParallelFactorPopMk=Plot.YSet}

Eventser.CurrentTimePointPopMk=Scheduler.TimePointerSet

Eventser.ContextPopMk=Scheduler.EventserContextSet

\\ Настройка Eventser-а

Scheduler.NCoresSet=30

Eventser.NEUSet=30

\\ Настройка ФУ вычислительного поля для теста Graph500

GraphGen.ContextPopMk=GraphCollector.GraphGenSet

GraphGen.ScaleSet=10

GraphGen.EdgeFactorSet=16

GraphGen.FUFieldRangeSet=7

\\ Перевод ФУ в режим моделирования

GraphGen.ManualModeSet=true

GraphCollector.ManualModeSet=true

\\ Запуск теста

GraphGen.Run=64

Console.Out="Средняя длина пути в графе: "

GraphCollector.TraceLongPopMk=Console.Out

Eventser.Start

\\ Вывод результатов моделирования

Console.LnOut="Process time: " Eventser.CurrentTimePointPopMk=Console.OutLn

Console.LnOut="Parallel dipertion: "Eventser.DispParallelFactorPopMk=Console.Out

Console.LnOut="Evens count: "Eventser.EventsCountPopMk=Console.Out

Console.LnOut="Use factor: " Eventser.UseFactorPopMk=Console.Out

*Листинг 4. Реализация теста бенчмарка Graph500 на ПЯ*

**4.3 Бенчмарк GREP – алгоритм сопоставления файловых строк шаблонам команды**

Grep – хорошо известная утилита командной строки различных операционных систем. С её помощью можно производить поиск по тексту строк, соответствующих некоторому заданному регулярному выражению.

Для проведения моделирования работы команды GREP необходим следующий набор функциональных устройств:

* String Sourсe
* Regexp Manager
* Regexp Worker
* Regexp Collector
* String Receiver

Структура виртуальной вычислительной системы ОА-архитектуры для реализации бенчмарка GREP приведена на рис. 70.



Рисунок 15 - Структура вычислительной системы для моделирования GREP

**ФУ для реализации бенчмарка команды GREP**

В Таблица 20 собран набор милликоманд ФУ, предназначенных для прогона тестового приложения бенчмарка команды GREP (алгоритма сопоставления файловых строк шаблонам команды GREP) на модели суперкомпьютерной системы ОА-архитектуры.

|  |  |  |
| --- | --- | --- |
| **Индекс милликоманды** | **Мнемоника милликоманды** | **Описание** |
| 0 | Reset | Сброс ФУ |
| 20 | PatternSet | Установка поискового шаблона (регулярное выражение) |
| 21 | StringSet | Установка входной строки для поиска |
| 22 | PatternSetStart | Установка поискового шаблона (регулярное выражение), затем запуск поиска |
| 23 | StringSetStart | Установка входной строки для поиска, затем запуск поиска |
| 25 | ResultMkSet | Установка милликоманды для отправки результата (первого найденного вхождения) |
| 26 | ResultMatrMkSet | Установка милликоманды для отправки результата в формате массива/матрицы (все внайденные вхождения) |
| 27 | ResultMatrPop | Выполнить милликоманду отправки результата в формате массива/матрицы |
| 28 | ResultMatrPopMk | Установить и выполнить милликоманду отправки результата в формате массива/матрицы |
| 29 | ResultProgSet | Установить миллипрограмму для отправки результата |
| 30 | Start | Запуск поиска |
| 41 | IndexOffsetSet | Установка стартового значения индекса при индексации входных строк. Используется в качестве милликоманды при приеме входной строки. (по умолчанию - 1000) |
| 42 | ResultFUContextSet | Установка контекста ФУ, которое будет принимать результаты поиска |

*Таблица 3 - Милликоманды устройства вычислений бенчмарка команды GREP.*

**Предварительная настройка функциональных устройств**

**ФУ String Source.** Данное функциональное устройство является поставщиком входных данных в программу моделирования программы GREP. Устройство производит построчное чтение некоторого заданного текстового файла и пересылка данных (строк) следующему функциональному устройству. Ниже приведен пример программы:

NewFU={Mnemo="StringSource" FUType=FUStringsSource}

…

StringSource.FileSet="eug3.txt"

StringSource.MkSet=RegManager.StringPut

*Листинг 5. Пример вызова ФУ StringSource*

**ФУ Regexp Worker.** Для проведения моделирования необходимо создать всего одно ФУ Regexp и провести его настройку. В качестве входных данных устройство принимает строку для поиска, а также поисковый шаблон, построенный по правилам регулярного выражения. После получения этих входных данных устройство произведет поиск искомой строки по шаблону и сформирует массив вхождений. Этот массив ФУ прикрепит в качестве нагрузки ко входной милликоманде и перенаправит её на контекст устройства, определенного в переменной контекста ResultFUContextSet.

После настройки данного ФУ Regexp Manager произведет его «клонирование», т.е. система будет состоять из нескольких одинаково настроенных ФУ Regexp Worker, что обеспечит распараллеливание решаемой задачи. Ниже представлен пример программы:

NewFU={Mnemo="Reg" FUType=FURegexp}

…

Reg.PatternSet="[A-Za-z]+"

Reg.ContextPopMk=RegManager.RegexpFUSet

*Листинг 6. Пример вызова ФУ RegexpWorker*

**ФУ Regexp Manager.** Данное устройство выполняет функции инициализации «рабочих» ФУ Regexp Worker, а также функции распредения входных данных (строк) между ФУ Regexp. Перед запуском данного устройства необходимо сначала настроить одно ФУ Regexp, а затем передать ссылку на его контекст устройству Regexp Manager через милликоманду RegexpFUSet. Необходимо также определить количество «клонированных» устройств через милликоманду CountSet. Данная милликоманда не только определит переменную контекста Count, но и создаст и проинициализирует устройства ФУ Regexp, используя предустановленный в переменной RegexpFU контекст оригинального устройства-обработчика. За приём входной строки отвечает милликоманда StringPut (входная строка в нагрузке). После получения всех необходимых данных устройство начинает слать милликоманды (начиная с 1000), номера которых соответствуют индексам входных строк, увеличенных на тысячу (+1000). Т.е., первая строка будет передана некоторому устройству Regexp Worker милликомандой 1000, вторая – 1001, третья – 1002 и т.д. Ниже представлен пример программы.

NewFU={Mnemo="RegManager" FUType=FURegexpManager}

…

Reg.ContextPopMk=RegManager.RegexpFUSet

RegManager.Clone=10

StringSource.MkSet=RegManager.StringPut

*Листинг 7. Пример вызова ФУ RegexpManager*

Для запуска тестовой задачи бенчмарка GREP на вычислительной системе ОА-архитектуры необходимо выполнить следующую миллипрограмму на ПЯ (ОА-языке):

NewFU={Mnemo="Scheduler" FUType=FUScheduler}

Scheduler.ContextPopMk=MainBus.SchedulerContextSet \\ Настройка ссылки на балансир (планировщик)

NewFU={Mnemo="StringSource" FUType=FUStringsSource}

NewFU={Mnemo="RegManager" FUType=FURegexpManager}

NewFU={Mnemo="Reg" FUType=FURegexp}

NewFU={Mnemo="RegCollector" FUType=FURegexpCollector}

NewFU={Mnemo="Eventser" FUType=FUEventser}

NewFU={Mnemo="Chart" FUType=FUChart}

NewFU={Mnemo="Plot" FUType=FUPlot}

\\ Настройка окна вывода графика

Plot.SeriaCreat

Chart.ParentSet

Chart.Caption="Parallel factor"

Chart.Focus

Plot.SeriaCreat

Chart.ChartPopMk=Plot.ParentSet

Plot.ColorSet=clRed

Plot.PenWidthSet=3

Chart.TitleYSet="Parallel factor"

Chart.TitleXSet="Model time"

Chart.TitleSet="Grep parallel factor"

Eventser.OutProgSet={Eventser.CurrentTimePopMk=Plot.XSet Eventser.CurrentParallelFactorPopMk=Plot.YSet}

Eventser.CurrentTimePointPopMk=Scheduler.TimePointerSet

Eventser.ContextPopMk=Scheduler.EventserContextSet

Scheduler.EventserMkSet=1 \\Милликоманда добавления события для Eventser

Scheduler.NCoresSet=9

Eventser.NEUSet=9

\\ Настройка эталонного ФУ Regexp

Reg.PatternSet="(день|ночь|она?|луна|мне|тебе|[0-9]+|[A-Za-z]+)"

Reg.ContextPopMk=RegManager.RegexpFUSet

RegCollector.ContextPopMk=Reg.ResultFUContextSet

\\ Создание вычислительного поля

Reg.ManualModeSet=true

RegManager.Clone=40

StringSource.FileSet="eug2.txt"

StringSource.MkSet=RegManager.StringPut

RegManager.ManualModeSet=true

\\ Настройка времени планирования вычислений

Scheduler.SheduleTimeSet=3000

\\ Запуск теста

StringSource.Start

Eventser.Start

\\ Вывода результатов тестаConsole.Out="Process time: " Eventser.CurrentTimePointPopMk=Console.OutLn

Console.Out="Process time: " Eventser.CurrentTimePointPopMk=Console.OutLn

Console.Out="Parallel factor: " Eventser.ParallelFactorPopMk=Console.OutLn

Console.Out="Use factor: " Eventser.UseFactorPopMk=Console.OutLn

*Листинг 8. Реализация теста Regexp по обработке текстовых данных*

На ПЯ также могут быть реализованы и миллипрограммы обработки результатов прогона тестовой задачи на ОА-архитектуры для их последующей визуализации и анализа. Например, миллипрограмма определения зависимости времени выполнения теста Grep от числа ФУ и ИУ будет выглядеть следующим образом.

NewFU={Mnemo="Scheduler" FUType=FUScheduler}

Scheduler.ContextPopMk=MainBus.SchedulerContextSet

NewFU={Mnemo="StringSource" FUType=FUStringsSource}

NewFU={Mnemo="RegManager" FUType=FURegexpManager}

NewFU={Mnemo="Reg" FUType=FURegexp}

NewFU={Mnemo="RegCollector" FUType=FURegexpCollector}

NewFU={Mnemo="Eventser" FUType=FUEventser}

NewFU={Mnemo="File" FUType=FUGatewayFile}

NewFU={Mnemo="Matr" FUType=FUConstMatr}

NewFU={Mnemo="GatewayFile" FUType=FUGatewayFile}

NewFU={Mnemo="Counter" FUType=FUCounter}

NewFU={Mnemo="Counter2" FUType=FUCounter}

Eventser.CurrentTimePointPopMk=Scheduler.TimePointerSet

Eventser.ContextPopMk=Scheduler.EventserContextSet

Scheduler.EventserMkSet=1 \\Милликоманда добавления события для Eventser

Scheduler.SheduleTimeSet=30

Reg.PatternSet="(день|ночь|она?|луна|мне|тебе|[0-9]+|[A-Za-z]+)"

Reg.ContextPopMk=RegManager.RegexpFUSet

RegCollector.ContextPopMk=Reg.ResultFUContextSet

Reg.ManualModeSet=true

RegManager.Clone=10

StringSource.FileSet="eug2.txt"

StringSource.MkSet=RegManager.StringPut

RegManager.ManualModeSet=true

\\ Создание матрицы для вывода результатов теста

Matr.RowSet=51 Matr.ColSet=51

Matr.IndexModeSet=0

Matr.Create

Scheduler.SheduleTimeSet=1000

\\ Цикл для сбора результатов

Counter.ValueSet=50

Counter2.ValueSet=50

Counter.LessEqProgSet={

Scheduler.ModelingReset

Eventser.ModelingReset

Counter2.PopMk=Scheduler.NCoresSet

Counter.PopMk=RegManager.Clone

StringSource.Start Eventser.Start

Counter.PopMk=Matr.RowSet Counter2.PopMk=Matr.ColSet

Eventser.CurrentTimePointPopMk=Matr.CellSet

Counter.Inc

}

Counter2.LessEqProgSet={Counter.Set=1 Counter2.Inc}

Counter2.Set=1

\\ Вывод результатов тестирования в файл

GatewayFile.FileNameSet="RegResult.txt"

GatewayFile.SeparatorSet=";"

GatewayFile.OpenFileWrite

Matr.MatrPopMk=GatewayFile.LnOut

GatewayFile.CloseFile

*Листинг 9. Реализация теста Regexp по обработке текстовых данных на ПЯ*

5. Проведение тестирования

Все проводимые ниже тесты были проведены для получения характеристик производительности моделируемой системы, описанных в разделе №2. Для каждого теста приведены полученные характеристики, а также различные зависимости некоторых выходных параметров от входных. Графики были получены при помощи среды MatLab.

**5.1 Тестовые прогоны на модели ОА-архитектуры dataflow-ВС задач расчета напряженности элетростатического поля**

Тестовые программы программы на ПЯ для задачи расчета напряженности в точках электростатического поля осуществлялись при изменении различных входных параметров моделируемой системы:

* Размерность матрицы вычислительных узлов (от 2\*2 до 40\*40 устройств)
* Расположение и числовое значение электрических зарядов на матрице
* Число исполнительных устройств в вычислительной системе

На графиках ниже приведены зависимости числа задействованных ядер от времени выполнения. Размер моделируемой матрицы – 70\*70 вычислительных узлов.

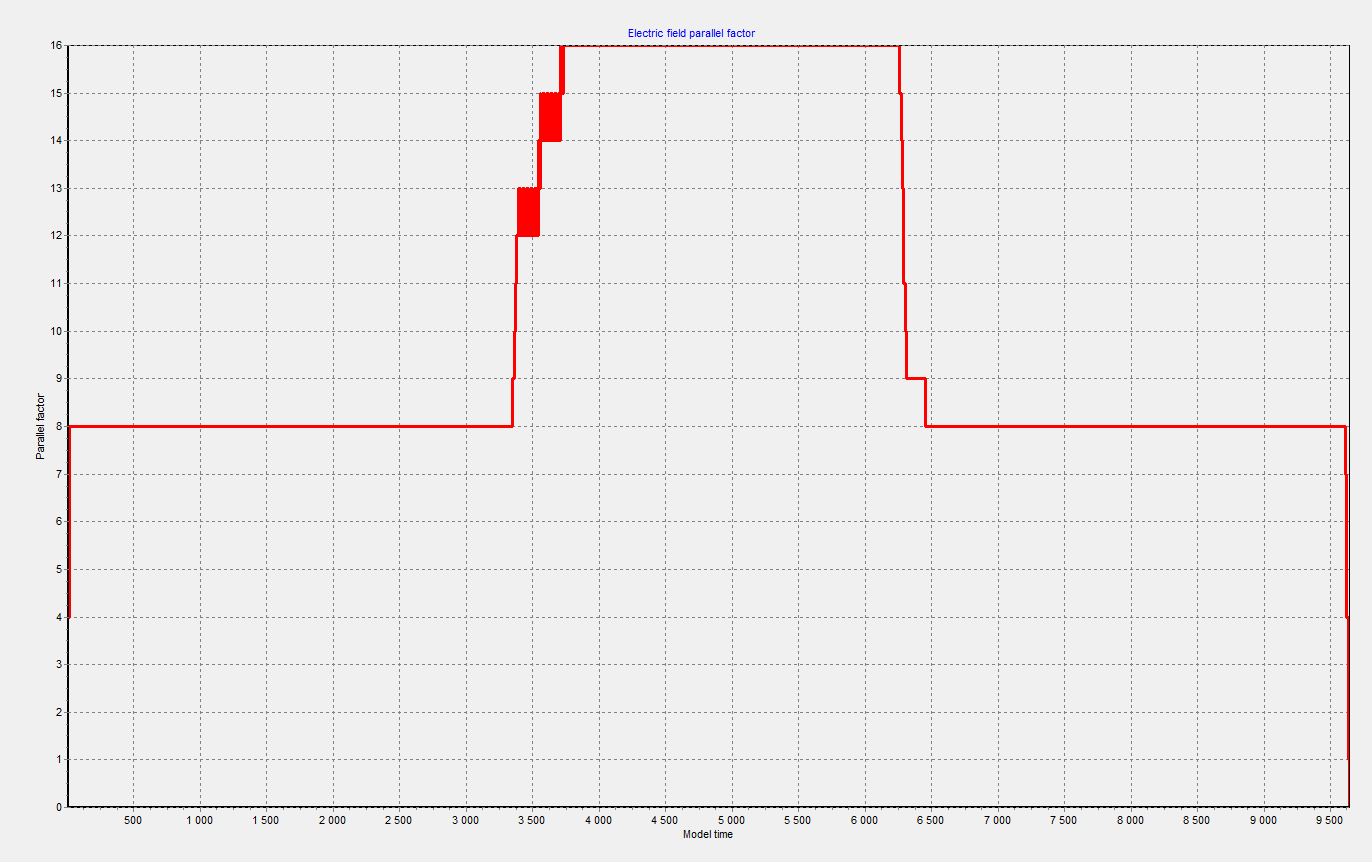
****

Рисунок 16. N=16, T=9611 т.

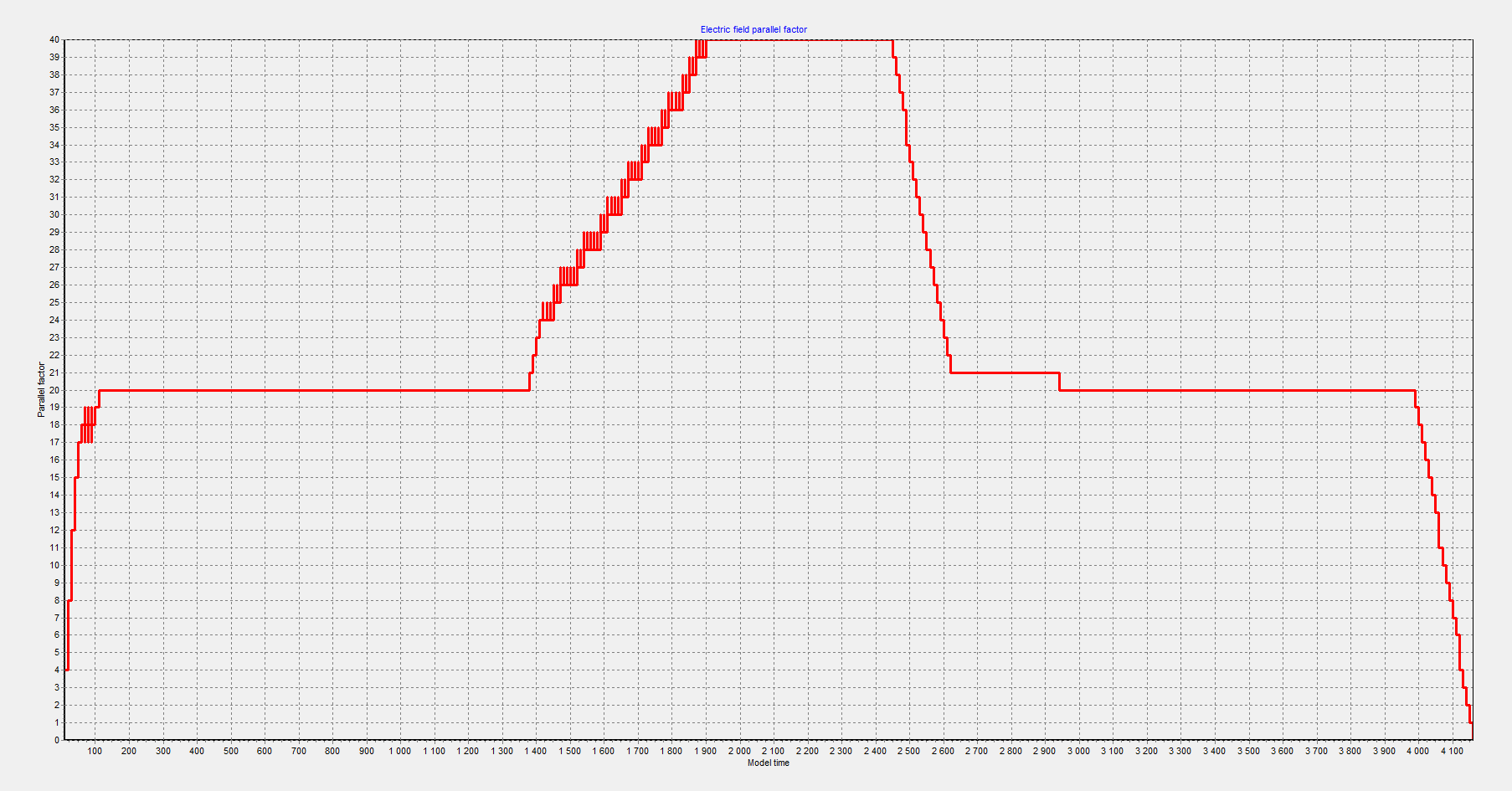


Рисунок 17. N=40, T=4115 т.

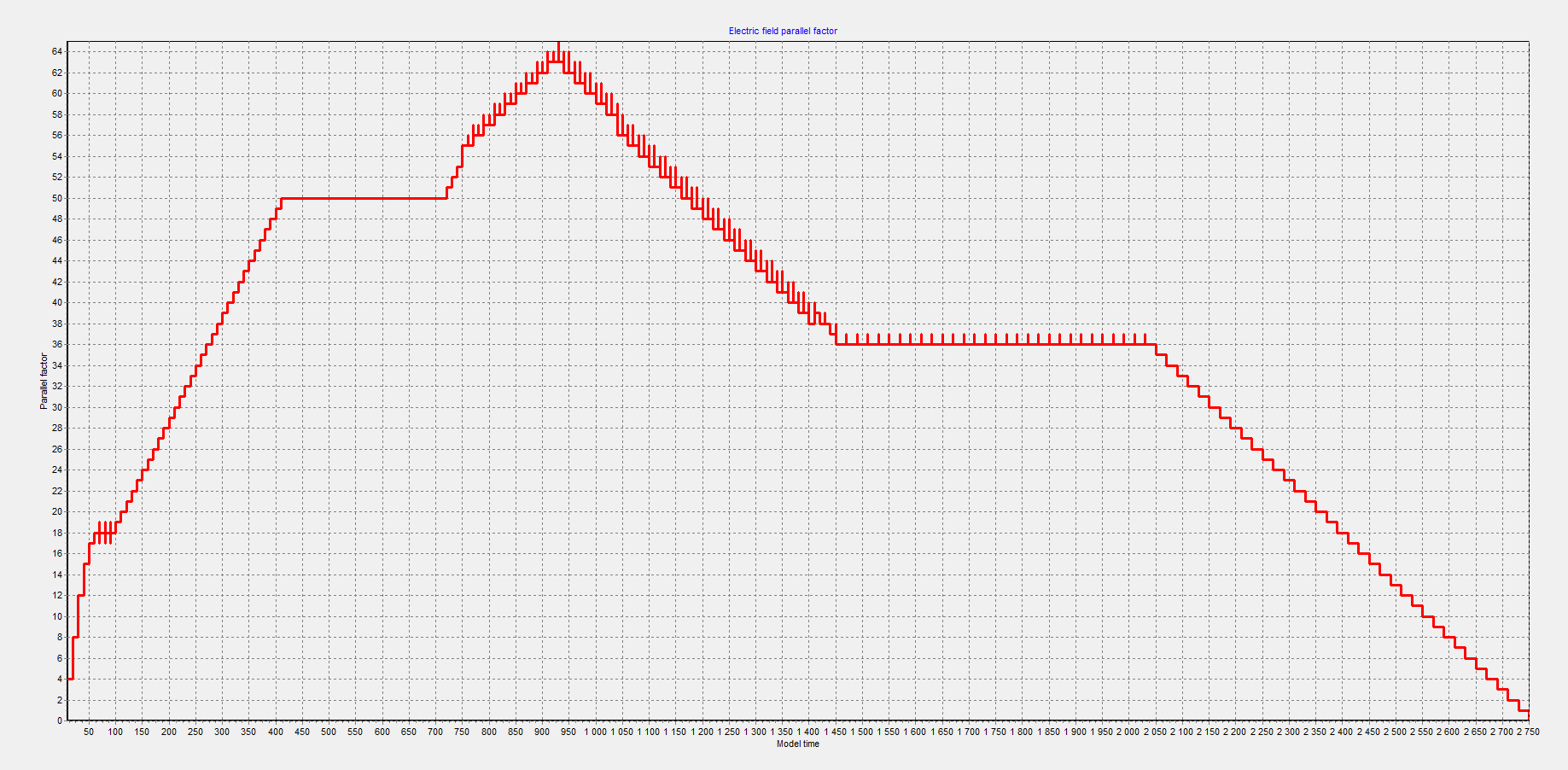


Рисунок 18. N=64, T=2750 т.

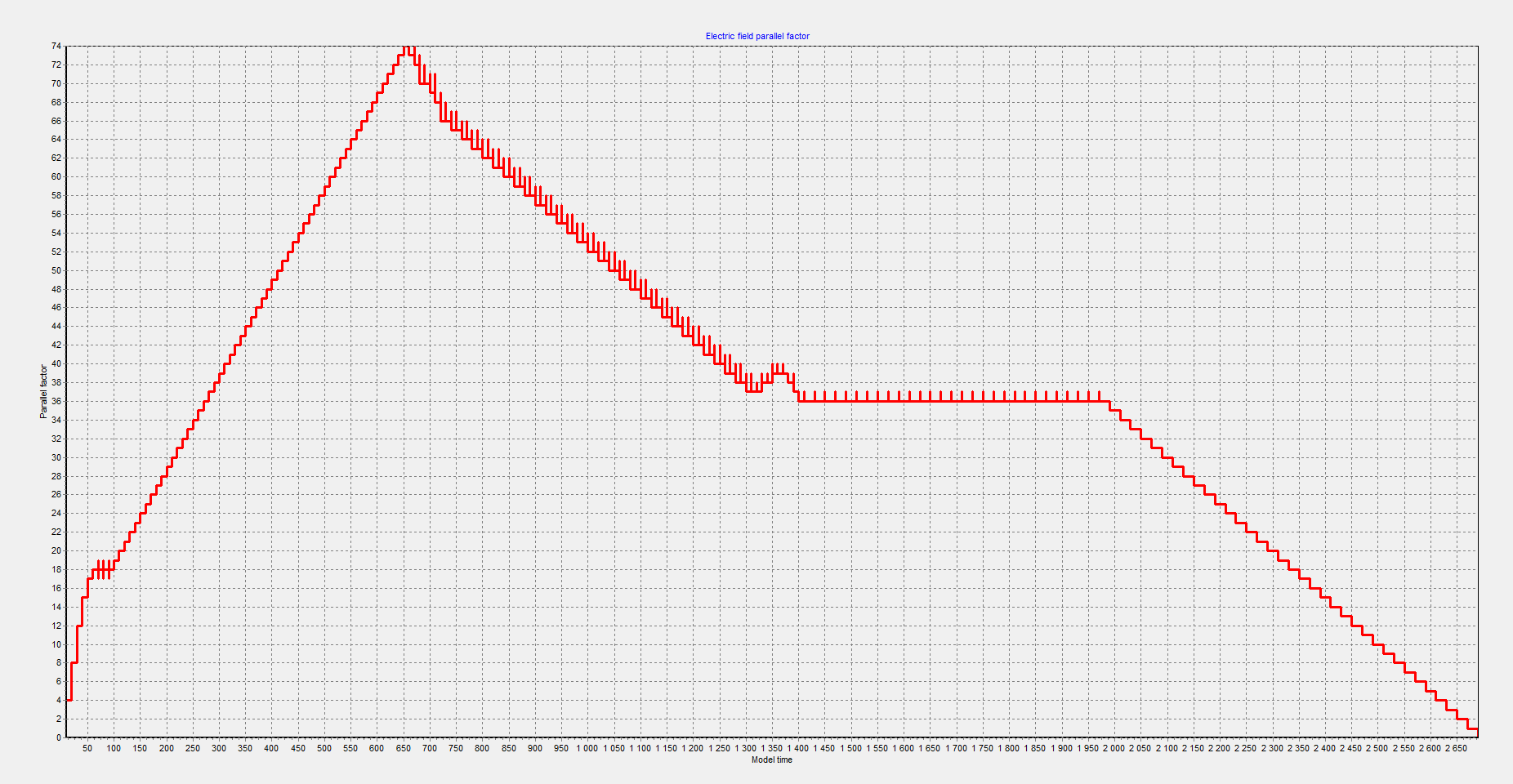


Рисунок 19. N=100, T=2691т.

Также, для тестовых задач расчета напряженности была разработана программа для визуализации электростатического поля. Градации серого цвета обозначают значение напряженности (белый – положительное, черный – отрицательный). Стрелки обозначают направление вектора напряженности.

На рисунке ниже приведен скриншот данной программы.

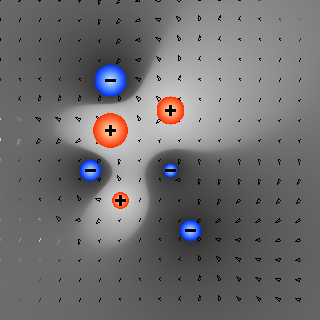
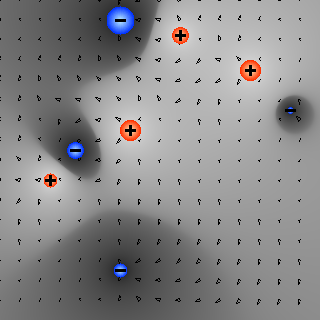
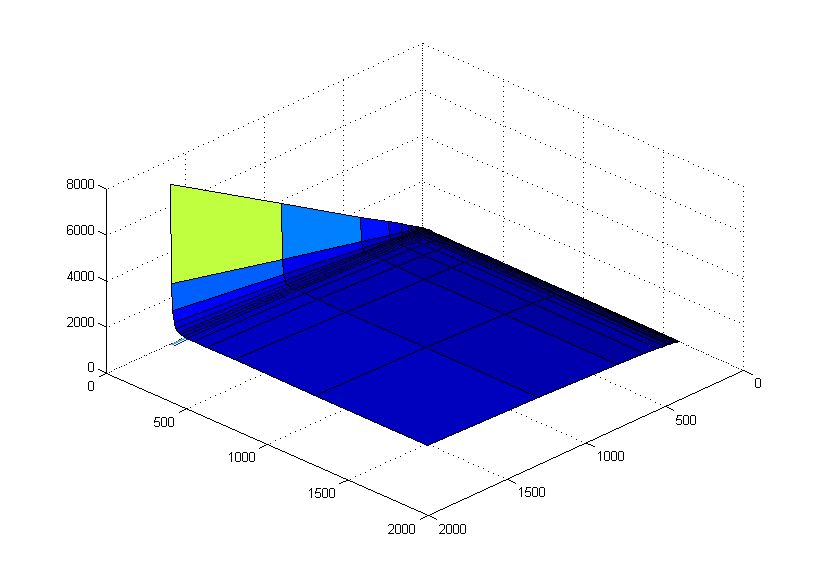
 

Рисунок 20, 21. Визуализация расчета напряженности электростатического поля.

На рисунках ниже приведен расчет выходных характеристик моделируемой вычислительной системы:

* зависимость модельного времени (T) выполнения теста от числа исполнительных ядер и от числа функциональных устройств
* зависимость среднего коэффициента параллелизма (P) от числа исполнительных ядер и от числа функциональных устройств
* зависимость дисперсии среднего коэффициента параллелизма (D) от числа исполнительных ядер и от числа функциональных устройств
* зависимость коэффициента использования аппаратуры (A) от числа исполнительных ядер и от числа функциональных устройств

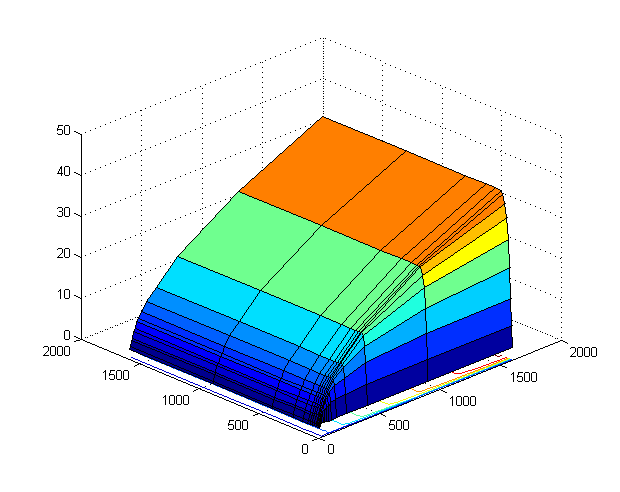


Rd

Fu

T

Рисунок 22. Зависимость времени выполнения (T) от числа исполнительных (Rd) и виртуальных устройств (Fu).



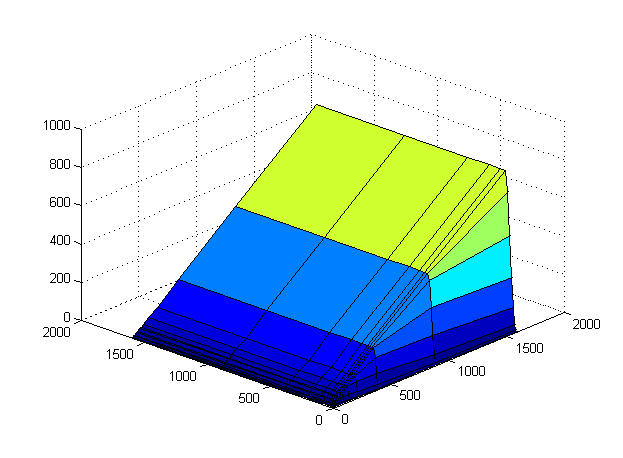
Fu

Rd

P

Рисунок 23. Зависимость коэффициента параллелизма P от числа исполнительных (Rd) и виртуальных устройств (Fu).

Производительность зависит только от числа функциональных устройств, растет по корню. Закон квадрата оборудования.



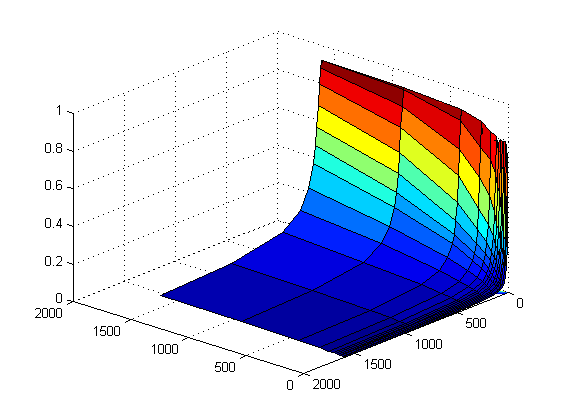
Rd

Fu

D

Рисунок 24. Зависимость дисперсии (D) среднего коэффициента параллелизма P от числа исполнительных (Rd) и виртуальных устройств (Fu).

A



Rd

Fu

Рисунок 25. Зависимость коэффициента использования оборудования (A) от числа исполнительных (Rd) и виртуальных устройств (Fu).

**5.2 Тестовые прогоны на модели ОА-архитектуры dataflow-ВС задач бенчмарка GRAPH500 на основе алгоритмов решения теоретико-графовых задач**

При прогонах тестовой программы, написанной на ПЯ, для алгоритма бенчмарка Graph500, изменялись значения входных параметров, таких как:

* N – число ИУ суперкомпьютерной системы при моделировании;
* 2M – число вершин графа
* K – число вершин графа, которое обрабатывает одно функциональное устройство

На следующих графиках результаты тестовых прогонов задачи бенчмарка Graph500 при различных значениях параметров N, M и K. Графики показывают зависимость коэффициента параллелизма вычислительной системы от времени выполнения теста.

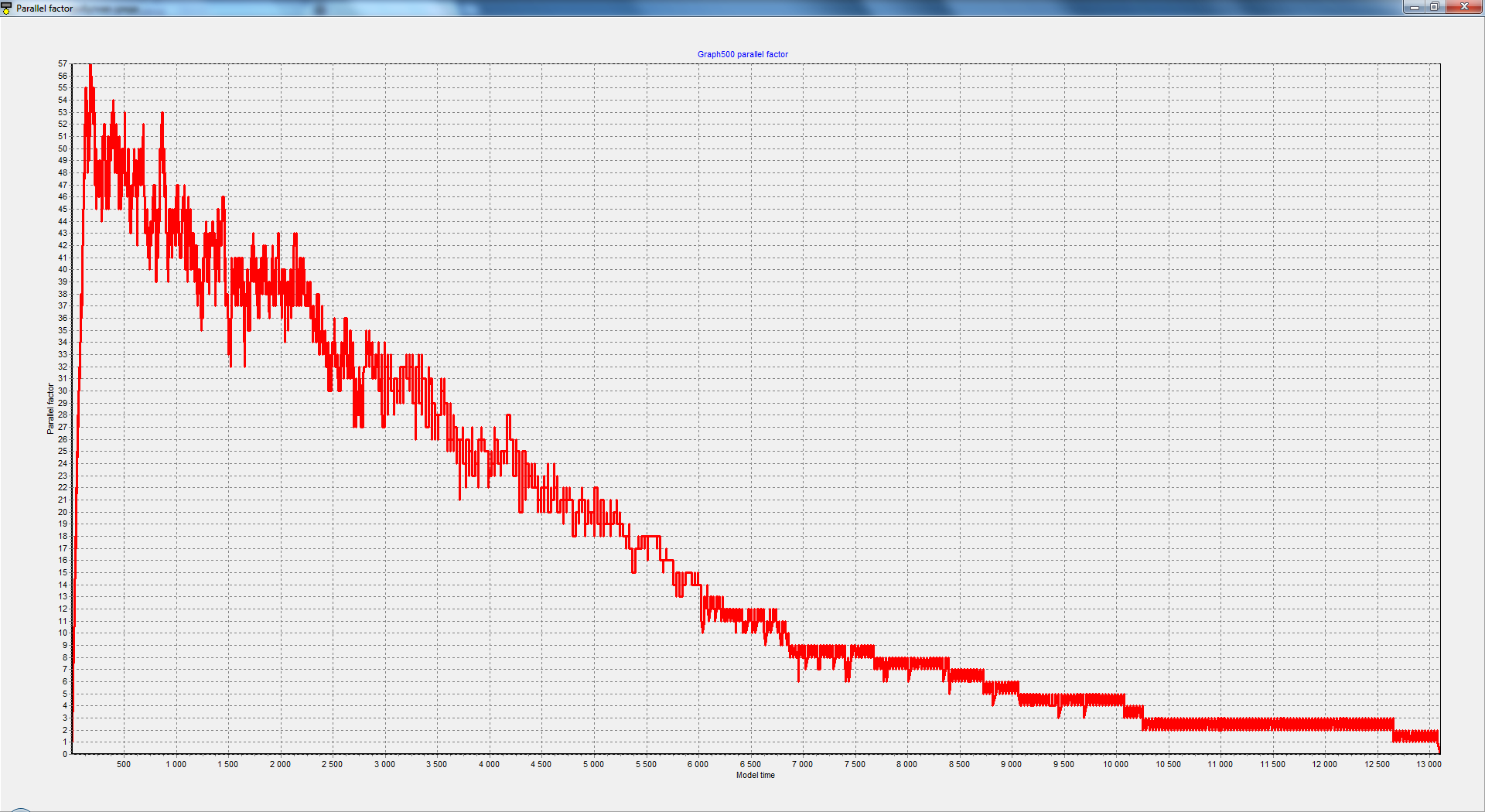


Рисунок 26 - Результаты моделирования теста Graph500 при N = 100 (максимально 57 ИУ задействовано), T = 13102, D = 228,768176408621.

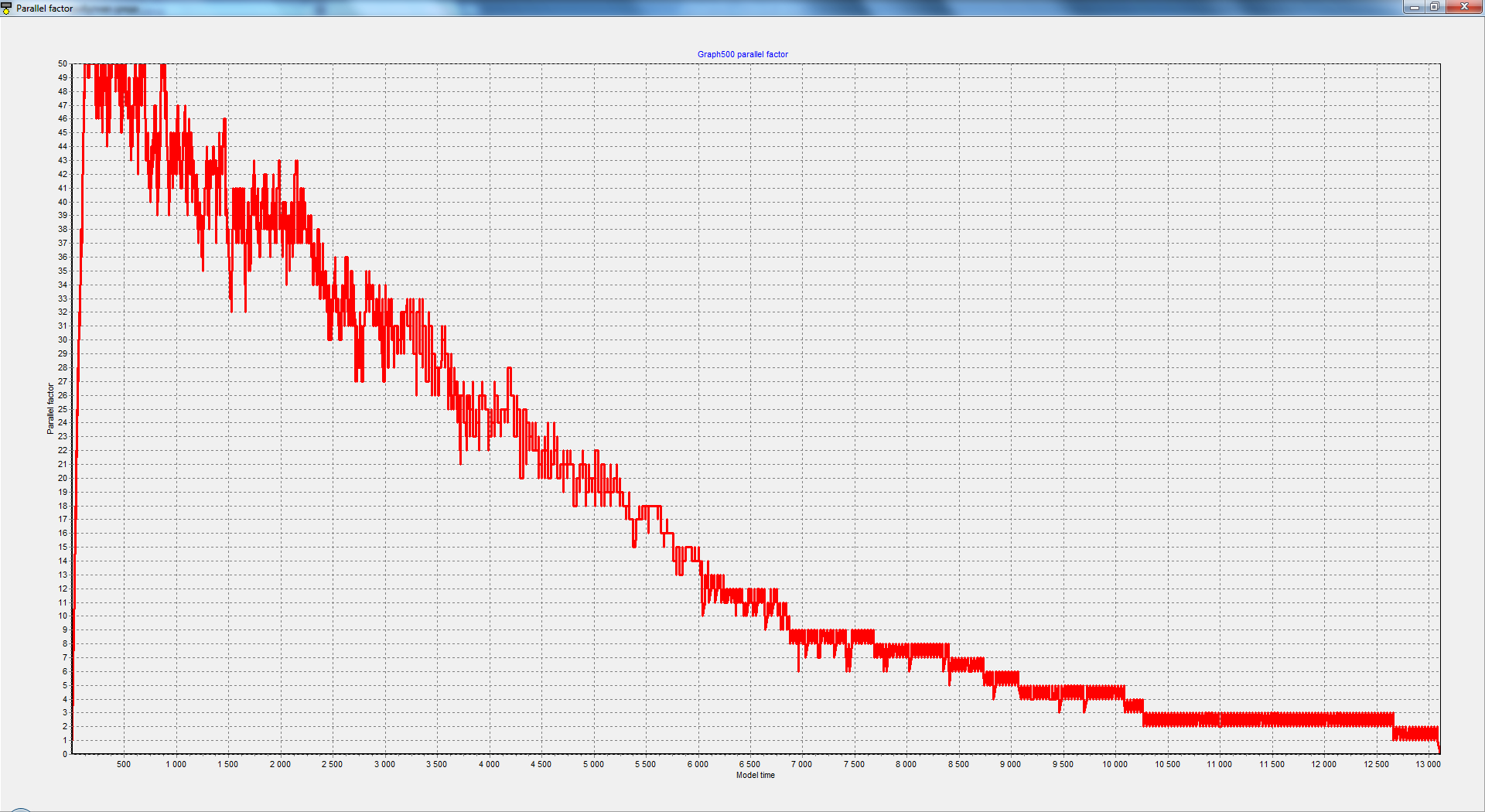


Рисунок 27 - Результаты моделирования теста Graph500 при N = 50, T = 13110, D = 227,085797409818

По результатам моделирования могут быть построены графики зависимости времени выполнения теста и дисперсии от числа ядер в dataflow-ВС (см. 9 и 86).

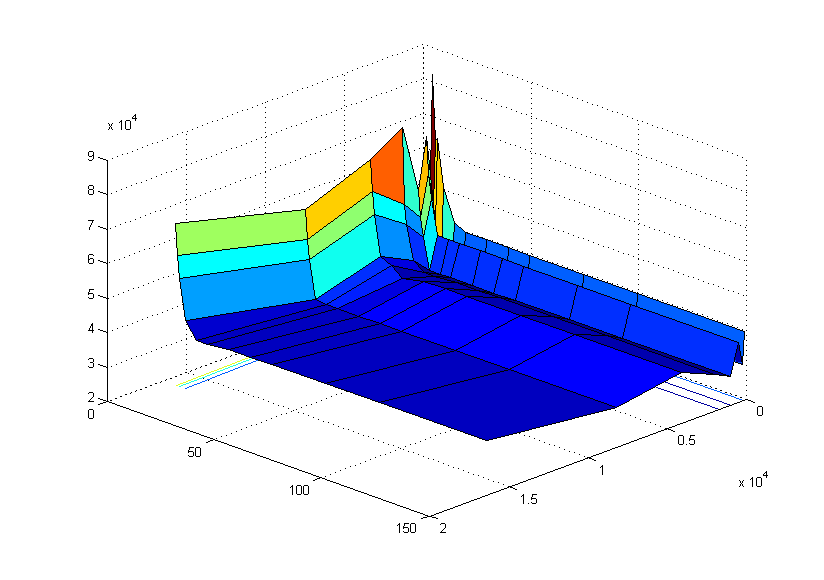
|  |  |  |
| --- | --- | --- |
| N | T | D |
| 1 | 221747 | 0 |
| 2 | 111097 | 0,004007322 |
| 5 | 46383 | 0,592867836 |
| 10 | 25890 | 7,814802056 |
| 15 | 19586 | 27,15531038 |
| 20 | 16642 | 58,18736677 |
| 30 | 14162 | 135,1538952 |
| 40 | 13302 | 200,9084474 |
| 50 | 13110 | 227,0857974 |
| 100 | 13102 | 228,7681764 |

Таблица 4. Зависимость времени выполнения и дисперсии теста Graph500 от числа ядер

Рисунок 28, 29 - Время выполнения и дисперсия от числа задействованных в ВП ИУ для теста Graph500

На рисунках ниже приведен расчет выходных характеристик моделируемой вычислительной системы:

* зависимость модельного времени (T) выполнения теста от числа исполнительных ядер и от числа функциональных устройств
* зависимость среднего коэффициента параллелизма (P) от числа исполнительных ядер и от числа функциональных устройств
* зависимость дисперсии среднего коэффициента параллелизма (D) от числа исполнительных ядер и от числа функциональных устройств
* зависимость коэффициента использования аппаратуры (A) от числа исполнительных ядер и от числа функциональных устройств

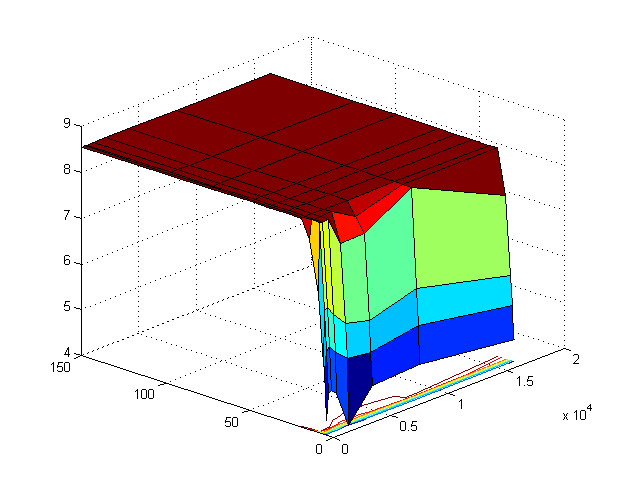


Rd

Fu

T

Рисунок 30. Зависимость времени выполнения (T) от числа исполнительных (Rd) и виртуальных устройств (Fu).

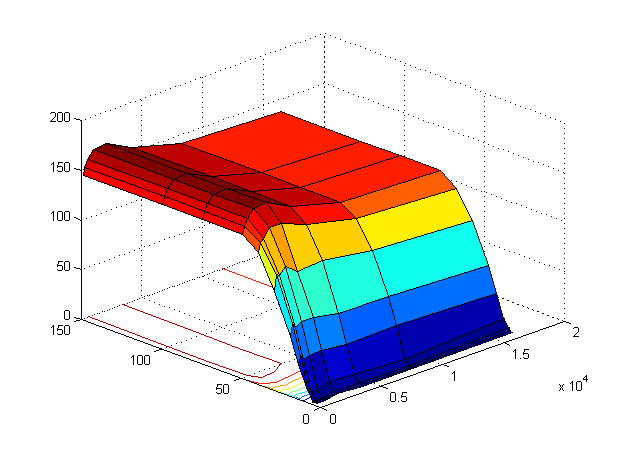


P

Rd

Fu

Рисунок 31. Зависимость коэффициента параллелизма P от числа исполнительных (Rd) и виртуальных устройств (Fu).

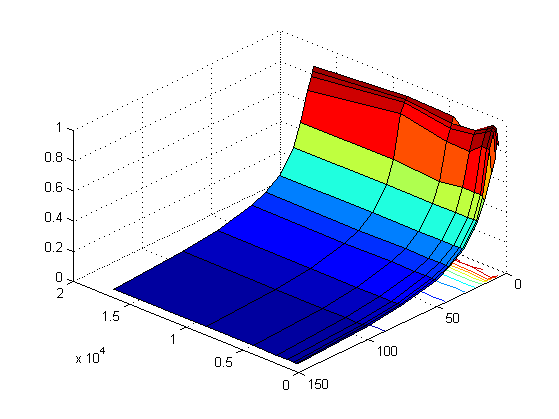


Rd

Fu

D

Рисунок 32. Зависимость дисперсии (D) среднего коэффициента параллелизма P от числа исполнительных (Rd) и виртуальных устройств (Fu).



Fu

Rd

A

Рисунок 33. Зависимость коэффициента использования оборудования (A) от числа исполнительных (Rd) и виртуальных устройств (Fu).

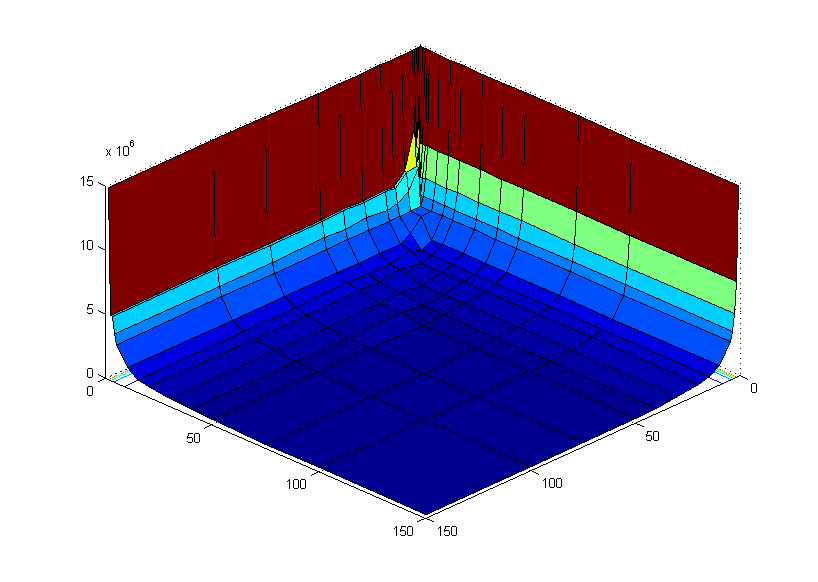
Наиболее оптимальное число ядер для данных параметров моделирования – 10-20 шт. По критерию «время исполнения (T)» оптимальное число – 15. По критерию «дисперсия (D)» - 20. После 20 ИУ время выполнения тестового набора Graph500 изменяется незначительно.

**5.3 Тестовые прогоны на модели ОА-архитектуры dataflow-ВС задач бенчмарка GREP – алгоритма сопоставления файловых строк шаблонам команды**

Ниже представлены результаты моделирования теста Grep на ОА-архитектуре. Моделирование проводилось при следующих параметрах анализируемого текста:

1. Число знаков – 18000;
2. Число строк – 1000.

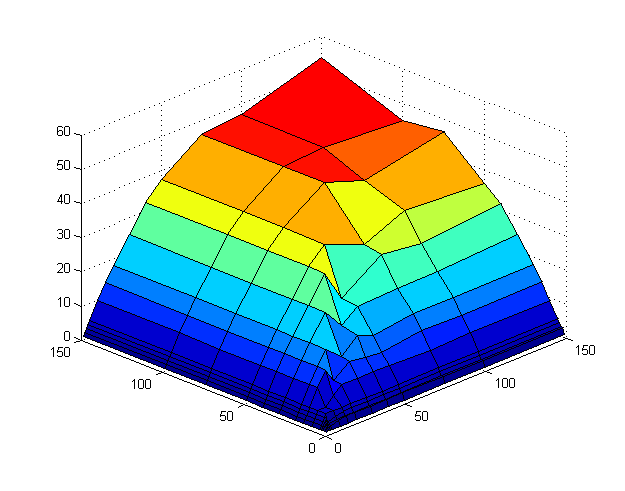
T



Rd

Fu

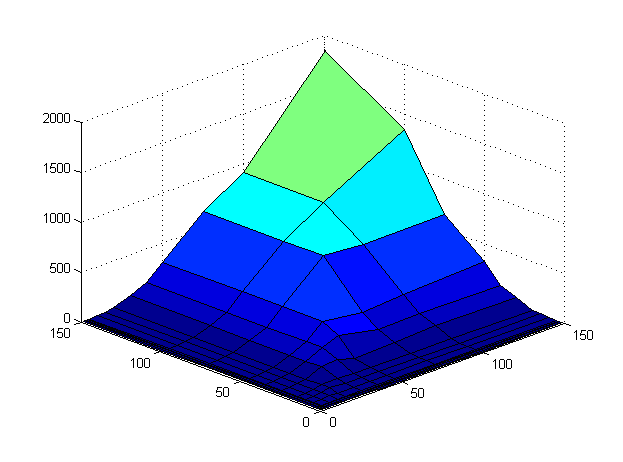
Рисунок 34. Зависимость времени выполнения (T) от числа исполнительных (Rd) и виртуальных устройств (Fu).

Рисунок 35. Зависимость коэффициента параллелизма P от числа исполнительных (Rd) и виртуальных устройств (Fu).

Rd

Fu

P

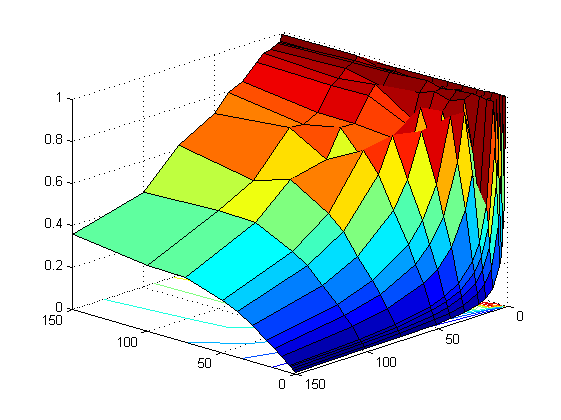
 Рисунок 36. Зависимость дисперсии (D) среднего коэффициента параллелизма от числа исполнительных (Rd) и виртуальных устройств (Fu).

A

Rd

D

Fu



Rd

Fu

Рисунок 37. Зависимость коэффициента использования оборудования (A) от числа исполнительных (Rd) и виртуальных устройств (Fu).

**5.4 Анализ результатов тестовых прогонов на модели ОА-архитектуры dataflow-ВС**

Запуск тестовых приложений на модели суперкомпьютера потребовал разработки индивидуальных способов реализации dataflow-архитектуры вычислительного процесса, настроенной на решение той или иной вычислительной задачи. В результате проведенных исследований было показано, что для максимально полного использования преимуществ dataflow-организации вычислений в параллельной суперкомпьютерной системе необходимо очень тщательно подходить к выбору «номенклатуры» (типов), «зернистости» (количества и размеров) и «качества» (функционального наполнения) ФУ архитектуры ВС. Набор ФУ ОА-архитектуры суперкомпьютерной системы может достаточно существенно различаться от задачи к задаче, что выдвигает на первый план задачу разработки эффективных методов адаптации архитектуры ВС к решению задач различной природы и топологии.

По имеющимся результатам тестов можно сделать вывод, что главным критерием эффективности системы является максимальное использование вычислительного оборудования, а также максимальное количество параллельно работающих устройств.

В задачах, для решения которых используются сеточные методы, анализируемая область разбивается на отдельные элементы, информационно связанные со своими «соседями». В этом случае для реализации вычислительной задачи на dataflow-ВС ОА-архитектуры каждый узел сетки оформляется в виде отдельного ВФУ, которое осуществляет вычисления и обменивается данными с помощью милликоманд со своими «соседями». По проведенным тестам, можно сделать вывод, что коэффициент использования аппаратуры зависит от числа «волн» вычислений. Иными словами, чем меньше устройств участвует в вычислении одной и той же «волны», тем ниже коэффициент использования аппаратуры.

В задачах, основанных на операциях с графами большой размерности мы также можем наблюдать незначительный прирост производительности, при повышении числа исполнительных устройств (ядер). Производительность теста в задаче вычисления наибольшего (или наименьшего) пути в графе сильно зависит от параметров задачи (число вершин, число связей между графами, число функциональных устройв и т.п.).

Наиболее высокие результаты тестирования видны в задаче поиска по тексту большого объёма. Это объясняется тем, что число «мини-задач», т.е. (число поиска вхождений в одной заданной строке) намного превышает число функциональных устройств. Алгоритм данной задачи построен таким образом, что весь объем текстовых данных (все строки) в самом начале работы программы распределяется между функциональными устройствами, таким образом, у каждого устройства сразу создаётся очередь задач на выполнение. При таком распределении время «простоя» некоторого устройства без задачи минимально, что является причиной высокой производительности ВС в данной задаче.

Для обеспечения максимально эффективной работы dataflow-ВС (с точки зрения использования доступных вычислительных ресурсов и достижения высокой степени параллелизма вычислений) требуется ее адаптация к решению конкретной вычислительной задачи, что подразумевает (по возможности автоматическую) подстройку ВП, представленного множеством ВФУ, под возможности имеющейся вычислительной среды. Критериями качества адаптации суперкомпьютерной ОА-системы к выполнению той или иной прикладной задачи может служить максимальный коэффициент использования оборудования.

Методы адаптации ОА-системы делятся на два класса:

- ручные, когда настройка конфигурации ВС и программы осуществляется программистом и инженером перед запуском ВП, включая выбор нужной топологии ВС, распределение сегментов расчетной сетки по вычислительным узлам, подбор оборудования с наиболее оптимальными характеристиками (число процессорных ядер в вычислительном узле, скорость передачи данных коммуникационным оборудованием и т.п.);

- автоматические, когда автоматическая корректировка вычислений (набор, количество и качество ВФУ) осуществляется непосредственно во время ВП.

При запуске и тестировании имитационной модели dataflow-ВС ОА-архитектуры варьировались следующие параметры ВС:

- количество вычислительных узлов в ВС;

- способ разбиения вычислительной сетки на сегменты

- топология вычислительной сети, объединяющей вычислительные узлы;

- скорость передачи данных по линиям связи;

- количество ИУ на вычислительном узле;

- количество ФУ на вычислительном узле.

При экспериментальных исследованиях критериями эффективности и качества работы ВС служил коэффициент использования оборудования (отношение количества задействованных в вычислениях ИУ к общему количеству ИУ), который должен быть максимальным.

Анализ ручной адаптации вычислительного процесса показал что, для достижения максимальной вычислительной эффективности ОА-системы необходимо соблюдение следующих правил по адаптации ВС к вычислительной задаче:

1. Среднее количество виртуальных функциональных устройств, задействованных в вычислениях, должно быть близко к числу ИУ, т.е. ядер системы. При такой конфигурации ВС тратит минимальное время на планирование вычислений, и ей требуется минимальный объем ОП. В том случае, когда NФУ < NИУ, коэффициент использования аппаратуры будет равен KИА ≈ NФУ; и при NФУ > NEU коэффициент использования KИА близок к единице, однако с дальнейшим увеличением числа функциональных устройств убывает. При NФУ = NИУ коэффициент использования оборудования KИА достигает своего максимума и примерно равен единице.
2. Пропускная способность канала (каналов) передачи данных должна быть больше поступающего потока данных из вычислительной сетки. В противном случае на ФУ «Шлюз» (ФУ, обеспечивающее передачу данных между вычислительными узлами) будет образовываться очередь из милликоманд, ожидающих передачу на соседний вычислительный узел. Милликоманды, ожидающие отправления, помещаются в буфер Шлюза и ожидают отправления. Зависимость объема буфера передаваемых милликоманд в ситуации, когда пропускная способность канала меньше поступающего из сегмента потока данных, выражается формулой: VMkBuf ~ (VМК \*NFUGateway - U)\*Nit\*Tit, где NFUGateway – число ФУ, передающих данные через Шлюз, VМК – объем памяти, занимаемый одной милликомандой, U – скорость передачи информации по каналу связи, Nit – номер итерации, Tit- время между итерациями (генерациями вычислительных волн).
3. Для задач с использованием сеточных методов средняя скорость прохождения (UCountWave) вычислительной волны должна быть примерно одинаковой во всех вычислительных узлах системы. Скорость прохождения вычислительной волны – это среднее время, за которое вычислительная волна продвигается по сетке на ФУ вперед (рис. 90). В случае, если в сегменте число требуемых для вычисления ФУ меньше или равно числу ИУ в вычислительном узле, данную величину можно вычислить по формуле: UCountWave=(ТWaveEnd - ТWaveStart)/(Lx+Ly), где ТWaveStart и ТWaveEnd – времена начала и конца вычислительной волны в сегменте сетки, Lx и Ly – размерность вычислительной сетки по x и y. В противном случае на данную величину могут влиять следующие факторы: среднее количество задействованных в вычислительной волне ФУ, количество ИУ в вычислительном узле, период генерации вычислительных волн, пропускная способность Шлюзов, осуществляющих передачу сгенерированных в сегменте данных на другие вычислительные узлы, время планировки вычислений (время выделения вычислительного ядра для ФУ). Для выполнения данного требования оператор и инженер, производящий конфигурацию ВС, должны оптимальным образом осуществить разбивку вычислительной сетки на сегменты и их распределение по вычислительным узлам, подобрать коммуникационное и вычислительное оборудование с наиболее подходящими параметрами.

Анализ результатов исследования методов адаптации суперкомпьютерной системы показал эффективность ручных методов адаптации. Также анализ выявил неэффективность автоматического анализа учета трафика (учета номера итерации) передаваемых через ФУ «Шлюз» милликоманд и эффективность учета номера итерации планировшиком (балансиром) во время планирования вычислений (ФУ, принявшее милликоманду с наименьшим номером итерации, помещается в начало очереди ожидания ресурсов) и эффективность метода информирования Шлюзом Топ-менеджера об опасности переполнения буфера передаваемых сообщений. Эффективность последнего способа снижается при увеличении размерности вычислительной сетки.

Выводы.

В диссертации были продемонстрированы преимущества применения dataflow-парадигмы в вычислительных системах. Были приведены различные программные реализации этой парадигмы.

ОА-архитектура, описанная в 1-м разделе, полностью реализует dataflow-подход к построению вычислительных систем. Было проведено математическое и имитационное моделирование данной архитектуры. Программной реализацией данной архитектуры является разработанная на кафедре ВСиС МИЭМ ОА-среда программирования и моделирования.

Были определены характеристики для оценки производительности данной архитектуры, а также разработаны тесты для получения этих характеристик. Тесты были реализованы и запущены в ОА-среде.

По результатам выполнения тестов были сделаны выводы о производительности и оптимальной структуре моделируемой ВС.

Список используемой литературы

* 1. Салибекян С.М., Панфилов П.Б., Моделирование суперкомпьютерной вычислительной системы объектно-атрибутной архитектуры с управлением потоком данных, Москва, 2012
  2. Салибекян С.М., Принципы милликомандной архитектуры как основа построения высокопроизводительных адаптивных вычислительных систем // Автоматизация и современные технологии. 2002. № 5
  3. 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>
  4. Салибекян С.М., Панфилов П.Б ОА-архитектура построения и моделирования распределенных систем автоматизации // Автоматизация в промышленности. N11, 2010
  5. Кельтон В., Лоу. А. Имитационное моделирование. Классика CS. 3-е изд. – СПб.: Питер; Киев: Издательская группа BHV, 2004
  6. Data flow computing: theory and practice / edited by John A. Sharp. Ablex Publishing Corp. Norwood, NJ, USA,1992.
  7. Модель акторов (<http://wiki.tza.su/wiki/Модель_акторов>)
  8. The Scala Language Specification Version 2.7 (<http://www-edlab.cs.umass.edu/cs530/ScalaReference.pdf> )
  9. 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>
  10. В. С. Бурцев. Выбор новой системы организации выполнения высокопараллельных вычислительных процессов, примеры возможных архитектурных решений построения суперЭВМ. // Сб. В.С. Бурцев Параллелизм вычислительных процессов и развитие архитектуры суперЭВМ М, 1997. (<http://www.computer-museum.ru/books/Burcev_parallelizm.pdf> )
  11. В. Н. Касьянов, Ю. В. Бирюкова, В. А. Евстигнеев. ФУНКЦИОНАЛЬНЫЙ ЯЗЫК SISAL 3.0 (<http://iis.nsc.ru/preprints/articles/pdf/sbor_kas_07_kasyanov_biryukova_evstigneev_sisal.pdf>)
  12. ALI R. HURSON, KRISHNA M. KAVI. DATAFLOW COMPUTERS: THEIR HISTORY AND FUTURE. (<http://www.csrl.unt.edu/~kavi/Research/encyclopedia-dataflow.pdf>)
  13. Карпов В.Э. Объектно-ориентированное программирование. Часть 1. Язык Смолток. Учебное пособие. - Московский государственный институт электроники и математики. М., 2000 — 45 с.
  14. Питерсон Дж. Теория сетей Петри и моделирование систем: Пер. с англ. - М.
  15. Салибекян С.М., Панфилов П.Б ОА-архитектура построения и моделирования распределенных систем автоматизации // Автоматизация в промышленности. N11, 2010.
  16. Салибекян С.М., Панфилов П.Б. Объектно-атрибутный подход к созданию интеллектуальных систем. IX Всероссийская научная конференция «Нейрокомпьютеры и их применение 2011». Тезисы доклада, М. 2011. С. 45. URL: <http://www.it.mgppu.ru/confnc/tezisy.pdf>
  17. Салибекян С.М., Панфилов П.Б. Объектно-атрибутный подход к построению интеллектуальных систем // Нейрокомпьютеры: разработки и применение. 2011, №11
  18. Салибекян С.М., Панфилов П.Б. Объектно-атрибутная архитектура – новый подход к созданию объектных систем // Информационные технологии. 2012, №2
  19. Баканов В.М., Потоковые (Dataflow) вычислители: управление интенсивностью обработки данных, Москва, 2010
  20. Graph500 Documentation, <http://www.graph500.org/>
  21. Салибекян С.М., Аминев Д.А., Семин В.Г., Результаты моделирования объектно-атрибутной вычислительной системы, 2012
  22. Салибекян С.М., Панфилов П.Б., Хакимуллин Е.Р., Гетерогенные вычислительные системы на основе объектно-атрибутной модели программирования
  23. [Ю.И. Рыжиков](http://www.books.ru/author/ryzhikov-32616/), Имитационное моделирование: Теория и технологии, Альтекс, Москва, 2004 г.