Перспективная суперкомпьютерная система на основе объектно-атрибутной модели вычислений с управлением потоком данных

Promising supercomputer which is based on dataflow object-attribute model

Салибекян С.М. (Salibekyan S.M.), Панфилов П.Б. (Panfilow P.B.)

Аннотация

Статья посвящена разработанной автором новой (объектно-атрибутной (ОА)) архитектуре вычислительной системы, работающей по принципу управления вычислениями с помощью потока данных (dataflow). Алгоритм в ОА-система задается с помощью описания обмена данными между функциональными устройствами. Вместо команд в ОА-системе используются милликоманды, которые представляют собой информационную пару, состоящую из нагрузки (данных или ссылки) и идентификатора.

Abstract

This paper presents a new architecture named object-attribute or OA. The OA-system is essentially a dataflow system. In the OA-system an algorithm is described with specification of information exchange among functional units (FUs). Instead of commands a millicommands are used in the OA-architecture. Millicommand consists of an attribute (tag) and a load (a constant or a reference to memory location).

Ключевые слова: объектно-атрибуная архитектура, милликоманда, управление вычислениями с помощью потока данных, параллельные вычисления, австракция данных и программного кода, изоморфизм программы и данных, искусственный интеллект, распределенная вычислительная система,суперкомпьютер, грид-технология.

Key words: Object-attribute architecture, millicommand, dataflow, parallel computing, program code and data abstraction, program and data isomorphism, machine intelligence, distribute computing system, supercomputer, grid technologies.

Введение

Принцип dataflow (управления вычислениями с помощью потока данных) является альтернативой общепринятому в настоящее время принципу управления потоком команд

(control flow) и считается чрезвычайно перспективным направлением развития вычислительной техники. Данная парадигма может реализовываться как на уровне аппаратуры, так и на уровне программы. Однако до сих пор не было разработано подходящей архитектуры вычислительной системы, которая могла бы конкурировать с control flow как на "железном", так и на программном уровнях.

Попытки создания вычислительных систем (ВС), работающих по принципу dataflow, начинаются с 1970-х годов (теоретические основы подобного класса ВС были заложены 1960-х годах, Карпом и Миллером), когда Джеку Деннису (Jack Dennis) удалось реализовать первый реальный образец dataflow-машины [1,2]. Джек Деннис и его последователи занимались реализацией dataflow-систем с памятью команд. Основная идея подобных компьютеров заключается в том, что команды классического фон неймановского формата помещаются не в оперативной памяти, а в специальной памяти команд: под каждую команду выделяется область памяти (шаблон), которая содержит код операции, поля операндов (могут быть и незаполненными), и адреса ячеек памяти, куда следует записывать результат выполнения команды. Если в ячейки операндов шаблона команды заполняются данными, то код операции и операнды передаются на один из фон неймановских процессорных элементов, что и производит соответствующие вычисления. Дальнейшее развитие концепции dataflow-систем с памятью команд – динамический dataflow, где в фишке (пакет, переносящий результат вычисления из процессорного элемента в память команд), добавляется поле, задающую итерацию цикла. Таким образом, удается распараллеливать вычисления нескольких итераций цикла. Следующий шаг в развитии данной концепции – системы с памятью фреймов: каждой итерации цикла выделяется отдельная область памяти, где хранятся все переменные данной итерации. Благодаря использованию памяти фреймов удается значительно упростить оборудование записи операндов в память команд [3].

Однако после 1990-х годов интерес к dataflow системам сошел на нет. Причиной тому послужил тот факт, что они значительно уступали по производительности классическим фон неймановским машинам, ввиду того, что вычислительным ядром подобных систем является фон неймановское вычислительное ядро, которое по определению на сможет работать быстрее фон неймановской машины. А оборудование dataflow систем, предназначенное для записи операндов, выборки команд из памяти команд для выполнения на процессорных элементов и т.д., является лишь «обременительной нагрузкой», значительно ухудшающей характеристики ВС.

В последнее время интерес к аппаратным dataflow системам возобновился благодаря появлению программируемых логических матриц (FPGA): dataflow системы «зашивается» в FPGA, т.е. практически реализуется аппаратно, благодаря чему и обеспечивается высокая

производительность. Однако подобные системы не обладают гибкостью и используются в основном в качестве специализированных сопроцессоров.

Весьма много реализаций концепция dataflow получила и на программном уровне [4]. Данный подход реализуется в объектно-ориентированном программировании (ООП), где программа представляет собой совокупность объектов, обменивающимися между собой сообщениями (вызов методов, принадлежащих классу). Однако данная концепция неудобна для реализации распределенных вычислений, т.к. объекты должны располагаться в едином адресном пространстве [5]. Более подходящей для параллельных систем является модель акторов [6,7], где программа представляет собой совокупность маленьких автономных программ (акторов), обменивающиеся между собой асинхронными сообщениями через специальные «почтовые ящики». В результате, акторы могут располагаться в различных адресных пространствах оперативной памяти (т.е. на различных вычислительных узлах), что дает возможность применения модели на несимметричных многопроцессорных ВС. Однако модель акторов реализует только крупнозернистый параллелизм, ведь иначе накладных расходы на организацию «почтового ящика» намного превысят аппаратные и временные расходы ВС на работу самих акторов.

Мелкозернистый же параллелизм был весьма удачно реализован В.М. Глушковым и В.А. Торгашевым в мультипроцессорах с динамической архитектурой (МДА) [8-9]. В основу МДА положена теория динамических автоматных сетей, где программа представляет собой не линейную последовательность команд, а совокупность связанных между собой линиями обмена данных автономных вычислительных узлов (конечных автоматов). Начальное состояние алгоритма задается перечнем вычислительных узлов и структурой сети (совокупность информационных связей между узлами) и алгоритмом работы каждого узла. Структура сети может быть описана с помощью специального языка программирования, либо с помощью графического языка. Во время вычислительного процесса меняются не только состояния узлов, но может трансформироваться и вся сеть: узлы порождают узлов-потомков, уничтожают их и меняют информационные связи. Решением поставленной перед автоматной сетью задачи является такое состояние сети, в котором она теряет способность к автотрансформации. В МДА распараллеливание алгоритма происходит автоматически без участия программиста: каждому узлу системы выделяется свой виртуальный процессор, а операционная система сама принимает решение о том, какому узлу и когда выделять аппаратные ресурсы (процессор и память). Однако данная архитектура несмотря на довольно удачные экспериментальные реализации распространения не получила.

Объектно-атрибутная (ОА) архитектура для суперЭВМ

В данной статье представлена модель высокопроизводительной вычислительной dataflow-системы (модель может одинаково эффективно применяться как к аппаратной, так и к программной части ВС), построенной по совершенной иной архитектуре, которая сможет обеспечить гибкость, производительность, параллелизм и распределенность вычислительной системы (ВС). Особенность предлагаемого нами объектно-атрибутного подхода [10-12] заключается в полном отказе от классической фон-неймановской архитектуры (концепции control flow). ОА-архитектура работает по принципу «динамический dataflow» и обеспечивает максимальный параллелизм вычислений (в том числе и самораспараллеливание вычислений), удобную реализацию на распределенных ВС, состоящих из вычислительных узлов различной архитектуры, масштабируемость ВС, а также легкую переносимость программы с одной аппаратной платформы на другую и имеет еще массу полезных качеств.

Такие возможности обеспечиваются благодаря тому, что алгоритм в ОА-архитектуре задается не как последовательность инструкций, а как описание обмена информацией между функциональными устройствами (ФУ), которые могут быть реализованы как программным, так и аппаратным способом на вычислительных узлах системы. ФУ, например, могут выполнять функции арифметико-логического устройства (АЛУ), функции ввода-вывода, контроля оперативной памяти, управления вычислительным процессом и т.д. (АО-архитектура несколько напоминает собой акторную модель, хотя и имеет значительные отличия). Предложенная архитектура является управляемой данным (dataflow), т.к. ФУ активизируется лишь в тот момент, когда к нему поступают все необходимые для вычисления данные. Управление же вычислительным процессом может осуществляться как централизованно, когда устройство управления запрашивает результаты вычислений у ФУ, так и децентрализованным, когда в логику работы ФУ закладывается алгоритм выдачи полученных данных для других ФУ. Далее следует намного рассказать и о самой ОА-архитектуре, которая зиждится на нескольких ключевых понятиях:

Информационная пара (ИП) (атрибутированные данные) – совокупность нагрузки (данных или ссылки на данные), и ярлыка (атрибута/уникального идентификатора), описывающего нагрузку. Указатель, хранящийся в нагрузке ИП, может ссылаться на информационные конструкции любой сложности (переменные, массивы, списки, другие ИП и т.д.). Тип данных, помещенных в нагрузке, определяется по ярлыку ИП.

Функциональное устройство – это виртуальное (реализованное программным способом) или реальное (реализованное аппаратно) устройство, осуществляющее обработку данных. ФУ имеет внутренние виртуальные регистры (набор регистров будем именовать контекстом ФУ),

используемые для хранения промежуточных данных и результатов вычислений, и может исполнять некий набор милликоманд, описываемый алгоритмом функционирования этого ВФУ.

Милликоманда (мК) – это ИП, где ярлык указывает ФУ-ву, каким образом следует обрабатывать прикрепленную к нему нагрузку (описывает нагрузку), и задает тип данных, хранящихся в нагрузке.

Шина данных-атрибута (ШДА) – виртуальная (или реальная) шина, по которой ФУ обмениваются милликомандами между собой (рис. 1). ШДА в виртуальной реализации представляет собой ФУ, специализирующийся на передаче данных между другими ФУ.

Капсула – это совокупность информационных пар, служащих для описания определенного объекта (с помощью капсулы обеспечивается абстракция данных). Каждая ИП, входящая в капсулу, задаёт один из критериев описываемого объекта. Например, на ОА-языке можно описать следующий объект:

Параллепипед{Длина=10 Высота=15 Ширина=20},

где {} – данные знаки задают начало и конец описания капсулы;

= - знак сопоставления атрибута и информационной нагрузки (перед «=» находится обозначение атрибута, после – нагрузки);

«Параллепипед» - имя информационной капсулы.

ОА-дерево абстракций (смысловой граф). В качестве нагрузки в ИП может выступать не только константа, но и ссылка на переменную, капсулу или иную информационную структуру, благодаря чему появляется возможность синтезировать смысловой ОА-граф (ОА-дерево абстракций), состоящий из множества капсул, связанных ссылками между собой (рис. 1). Смысловой ОА-граф служит для описания сколь угодно сложного объекта. Капсула, входящая в состав ОА-дерева, может содержать не только данные, но и программу (миллипрограмму), которая состоит из последовательности милликоманд, помещенных в капсулу - таким образом, ОА-дерево превращается в мощную базу знаний, способную как распознавать объекты, так и управлять объектами «внешнего мира». ОА-дерево абстракций несколько напоминает собой структуру класса в объектно-ориентированном программировании (ООП), однако ОА-подход обладает намного большей гибкостью, т.к. механизм капсул и ссылок в качестве нагрузок ИП позволяет легко синтезировать и перестраивать описание смысловых конструкций уже во время выполнения вычислительного процесса по заранее заложенным программистом (или экспертом) правилам – информационные структуры, образуются путем группировки ИП в капсулы и настройки ссылок между капсулами (синтез ОА-дерева абстракций) (рис. 1). Вычислениями и синтезом ОА-дерева абстракций в ОА-арихтектуре занимаются ФУ. ОА-дерево абстракций несколько напоминает собой структуру класса в объектно-ориентированном программировании (ООП), однако ОА-подход обладает намного большей гибкостью, т.к.

механизм капсул и ссылок в качестве нагрузок ИП позволяет легко синтезировать и перестраивать описание смысловых конструкций уже во время выполнения вычислительного процесса по заранее заложенным программистом (или экспертом) правилам – информационные структуры, образуются путем группировки ИП в капсулы и настройки ссылок между капсулами (синтез ОА-дерева абстракций) (рис. 1). Вычислениями и синтезом ОА-дерева абстракций в ОА-арихтектуре занимаются ФУ.

Абстракция, реализованная в ОА-архитектуре, более выигрышна по сравнению с ООП. ОА-дерево абстракций хоть и несколько напоминает собой структуру классов в ООП, однако имеет ряд существенных преимуществ. Во-первых, синтез и модификация ОА-констоукций осуществляются непосредственно во время выполнения вычислительного процесса (в ООП, как правило, синтез классов происходит во время компиляции программы). Во-вторых, миллипрограмма представляет собой не что иное, как информационную капсулу, которая содержит в себе милликоманды, т.е. форматы информационной капсулы, хранящей в себе данные, и капсулы, хранящей миллипрограмму, совпадают. Это дает возможность встраивать капсулы с миллипрограммой в ОА-дерево абстракций. В-третьих, благодаря изоморфизму, обеспеченному ОА-архитектурой, появляется возможность легкой модификации ОА-деревьев во время выполнения вычислительного процесса. В-четвертых, обеспечивается синтез смысловых конструкций от простого к сложному, что совпадает со способом мышления человека. На синтезе смысловых конструкций (ОА-деревьев) и остановимся более подробно.

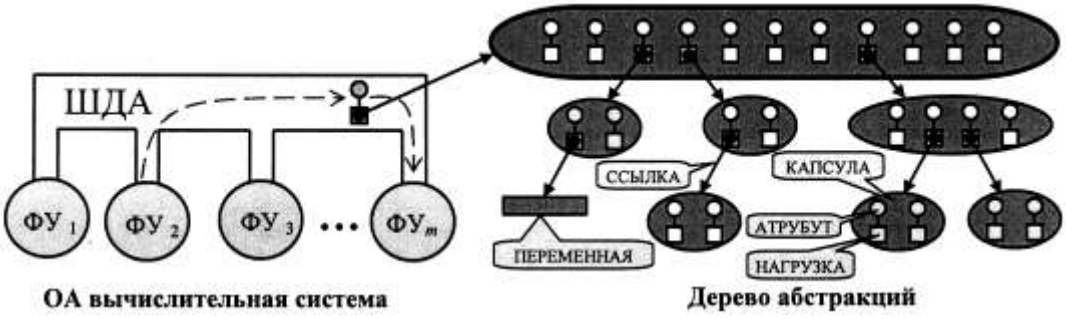

Рис. 1. Работа ОА-системы с деревом абстракций

Процесс синтеза информационных конструкций начинается с так называемых информационных атомов – элементарных (неразложимых) абстракций (атом в большинстве случаев представляет собой единственную ИП). Например, для систем распознавания изображений атом – описание отдельного пикселя изображения (координаты и код цвета). Для систем смыслового анализа текста атомами будут являться отдельные символы, составляющие анализируемый текст (рис. 2). Например, при смысловом анализе текста на первом уровне абстракции соответствующие ФУ проводят лексический анализ поступающих символов (символ является информационным атомом), т.е. формирование из символов слов. Далее

выделенные слова передаются на следующий уровень абстракции, где уже другие ФУ проводят синтаксический анализ поступивших слов. Информационные конструкции, сформированные на этом уровне, передаются на уровень выше для смыслового анализа и т.д. На каждом последующем уровне информационные конструкции (абстракции) становятся все более и более сложными, и их количество уменьшается (сложная абстракция представляет собой довольно большое смысловое ОА-дерево). На вершине этого так называемого конуса абстракций находится ключевая абстракция, описывающая распознаваемый объект полностью. Если же необходимо из общей абстракции произвести синтез более простых информационных конструкций, то применяется обратный конус абстракций (т.е. синтез от сложного к простому). Так, если требуется перевести текст с одного языка на другой, то используются сразу и прямой (распознание смысла исходного языка), и обратный (для синтеза текста уже на другом языке из ключевой абстракции) конусы (рис. 2). ОА-абстракция также обеспечивает удобную декомпозицию задачи, обеспечивает интероперабельность программы, что является весьма полезным свойством для сложнейшего программного обеспечения суперкомпьютеров.

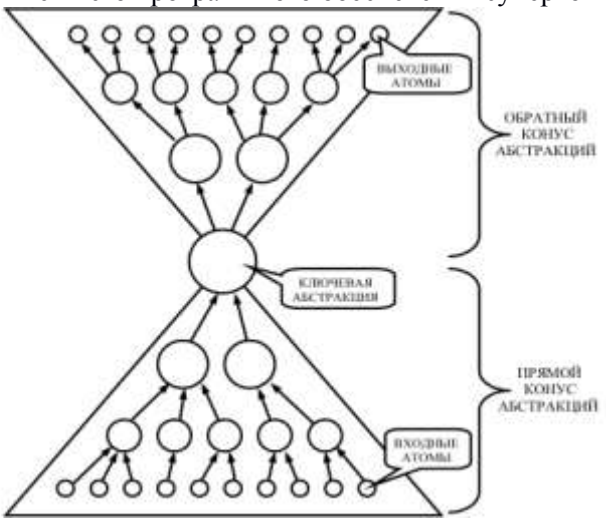

Рис. 2. Прямой и обратный конусы ОА-абстракций

В программной реализации ФУ представляют собой подпрограмму со стандартным интерфейсом, состоящим из трех полей. Например, функция реализации логики работы целочисленного арифметико-логического устройства, написанная на языке Си, будет выглядеть следующим образом:

```
void ALU(void *Context, int millicomand, void *Load )
{ switch(millicomand)
 {case 0: // Сброс (Reset)// в скобках мнемоника //милликоманды
   ((AluContext*)Context)->Accumulator=0;
  case 1: // Установить значение в аккумуляторе (Set)
            if(Load==NULL) ((AluContext*)Context)-> Accumulator=0;
```

```
            else ((AluContext*)Context)-> Accumulator=*PVariant(Load );
  case 2: // Сложение (Add)
if (Load !=NULL)
 {((AluContext*)Context)->Accumulator=
  ((AluContext*)Context)->Accumulator+*PVariant(Load );
  if(((AluContext*)Context)->Accumulator>=0)
    ((AluContext*)Context)->FLager=true;
 else ((AluContext*)Context)->FLager=false;
 if (((AluContext*)Context)->Accumulator==0)
   ((AluContext*)Context)->FZero=true;
else ((AluContext*)Context)->FZero=false;}
  case 4: // Выдать содержимое аккумулятора (Pop)
if (Load!=NULL) *PVariant(Load) = ( (AluContext*) Context)->Accumulator;}
// и т.д.……….……}
```

где *Context* – ссылка на контекст виотуального устройства;

millicomand – индекс милликоманды (каждая милликоманда имеет свой уникальный идентификатор);

Load – ссылка на нагрузку милликоманды (ссылка на информационную конструкцию, которая выступает в качестве нагрузки к милликоманде;

AluContext – описание структуры контекста ФУ АЛУ (имя структуры, где описаны регистры ФУ).

Контекст же ФУ целочисленное АЛУ можно описать с помощью следующей структуры данных:

```
struct AluContext{int Accumulator; //Аккумулятор
      bool FZero, FLager; } // Флаги нуля и знака результата
```

Далее, когда алгоритм реализации логики работы ФУ "прошит" в ОА-платформу, ФУ можно управлять с помощью ОА-образа. Например, ФУ для операция сложения на ОА-языке описывается следующим образом:

Bus{ALU.Set=2, ALU.Add=2, ALU.Pop=x}

где *Bus* – указание на то, что последовательность милликоманд, расположенных в информационной капсуле, должна быть выдана на Шину (ШДА);

ALU – имя ФУ;

ALU.Set, ALU.Add, ALU.Pop – расширенные милликоманды для устройства АЛУ (в расширенной милликоманде указывается ФУ, что должно исполнять милликоманду: перед точкой стоит обозначение ФУ, которому передается милликоманда, после точки – ярлык передаваемой нагрузки): *ALU.Set* – записать число из нагрузки милликоманды в аккумулятор

ФУ АЛУ, *ALU.Add* - сложить значение из аккумулятора с числом из нагрузки милликоманды и записать результат вычисления обратно в аккумулятор, *ALU.Pop* – выдать значение из аккумулятора и записать его в ячейку памяти с адресом, хранящимся в нагрузке милликоманды;

x – мнемоника ячейка памяти, куда АЛУ выдает результат своих вычислений.

Отличительные особенности ОА-архитектуры

После вышеприведенного введения в ОА-арихтектуру, сразу же можно отметить ее отличия от акторной модели:

- любой ФУ может передать данные любому другому ФУ, входящему в состав ВС;

- наличие в составе ОА-системы устройства (или нескольких устройств) Шина;

- у всех программ реализации логики работы виртуальных ФУ имеется одинаковый интерфейс, состоящий из трех полей (указателя на контекст ФУ, атрибута милликоманды и указателя на нагрузку милликоманды); такое решение унифицирует всю вычислительную систему, значительно облегчая ее модификацию, сопряжение различных частей программы и т.д.;

- тип и назначение пришедших по Шине данных ФУ определяет по ярлыку милликоманды, в результате чего отпадает необходимость описывать интерфейс ФУ, как это делается в акторной модели;

- возможность составлять информационные ОА-конструкции (смысловыве ОА-графы);

- более рациональное использование программного кода (для реализации алгоритма работы одного класса ФУ используется одна подпрограмма, которая работает с различными контекстами ФУ);

- ФУ Шина работает в последовательном режиме, но остальные ФУ могут работать в параллельном режиме (для ускорения обмена информацией в ВС могут применяться несколько Шин);

- ОА-вычислительную систему можно разделить на две части: ОА-платформа (совокупность подпрограмм реализации логики работы ФУ) зависящая от архитектуры вычислительного узла и ОА-образ независящий от архитектуры вычислительного узла (служит для описания обмена информацией (милликомандами) между ФУ); для того, чтобы перенести ОА-системы на вычислительный узел другой архитектуры, необходимо лишь реализовать ОА-платформу для новой архитектуры вычислительного узла (переписать, например, на языке высокого уровня подпрограммы реализации логики работы ВФУ);

- для описания ОА-образа используется специализированный язык программирования (ОА-язык), описывающий как параллельный алгоритм, так и структуры данных;

- ОА-образ может модифицироваться во время вычислительного процесса, т.е. без перекомпиляции программы (подобным обладает язык программирования Smaltalk, который называют единственным в мире истинно объектно-ориентированным языком).

ОА-архитектура для реализации суперкомпьютеров и грид-систем

Предлагаемая архитектура весьма удобна для реализации супрекомпьютерных ВС и грид-технологий. Причин тому несколько.

Во-первых, ОА-архитектура обладает "врожденным" параллелизмом: все ФУ представляются изолированными устройствами, обменивающимися данными через Шину, и потому они могут работать параллельно (как нить вычислений, так и тяжеловесный вычислительный процесс, так и работа различных вычислительных узлов, входящих в компьютерную сеть). Причем, одинаково эффективно реализуется как мелкозернистый, так и крупнозернистый параллелизм. АО-система способна не только обеспечивать эффективное распараллеливание, но также и осуществлять самораспареллеливание вычислений (самораспараллеливание обеспечивается благодаря концепции dataflow, где вычислительные операции инициируются приходом данных). Узким местом ОА-системы является Шина, работающая в последовательном режиме, однако это ограничение можно обойти применив в одной ВС несколько Шин (таким образом ВС разбивается на несколько информационных сегментов, которые также могут обмениваться информацией между собой через специальные ФУ; в результате, информационный трафик обмена между ФУ разбивается на несколько частей и нагрузка на Шины становится меньше.

Во-вторых, ОА-архитектура обладает и ещё одним весьма важным для суперкомпьютерных систем свойством: распределенное управление вычислительным процессом, когда вычислениями (как на крупнозернистом, так и мелкозернистом уровне) управляют сами ФУ (т.е. в системе на любом уровне отсутствует централизованное устройство, управляющее вычислениями). Причем, программист пишет программу для распределенной вычислительной системы как единое целое, а не создает отдельные модули и затем описывает обмен данными между ними.

В-третьих, предложенная архитектура обладает весьма простым интерфейсом передачи данных по линиям связи, что очень удобно для реализации распределенных ВС: формат передачи данных в ОА-системе еднообразен (передаются ярлык милликоманды и ее нагрузка (константа или информационная ОА-конструкция). В современных же распределенных системах протокол обмена информации чрезвычайно сложен: DCOM и Corba используют специальный язык IDL, предназначенный только для описания интерфейса удаленных процедур; протокол MPI также весьма громоздок.

В-четвертых, совокупность виртуальных ФУ, запущенных на выполнение в распределенной ВС, создает единую вычислительную среду, которая не зависит от аппаратной архитектуры входящих в ВС вычислительных узлов (архитектурно зависимой является лишь ОА-платформа, ОА-образ от архитектуры вычислительного узла не зависит). В результате появляется возможность создания гетерогенных вычислительных комплексов.

В-пятых, ВС ОА-архитектуры обладает свойством масштабируемости: виртуальные ФУ можно перемещать с одного вычислительного узла системы на другой, при этом не нарушается алгоритм работы ОА-программы в целом. Виртуальные ФУ могут создаваться, уничтожаться и перемещаться на другие вычислительные узлы и непосредственно во время вычислительного процесса. (Подобным свойством обладает МДА, где вычислительным узлам динамически выделяются аппаратные ресурсы [8-9].)

В-шестых, ОА-архитектура обладает свойством изоморфизма, когда изменение в части программы не влечёт за собой появления ошибок во всей программе. Изоморфизм поддерживается как на уровне структур данных, так и на уровне программы: благодаря нескольким нехитрым приемам можно безболезненно изменять как структуры данных, так программы реализации логики работы ФУ, так и сам ОА-образ.

В-седьмых, обеспечивается удобная технология создания программ: удачная абстракция и декомпозиция данных и программы (ФУ и смысловые ОА-конструкции); легкая совместимость различных "кусков" программы (универсальный интерфейс программы реализации логики работы ФУ и информационные ОА-конструкции как раз и обеспечивает унификацию интерфейса взаимодействия между программными модулями). ОА-конус абстракций позволяет весьма просто применять как восходящую, так и нисходящую технологию создания программы.

В-восьмых, ОА-архитектура идеально подходит для построения систем искусственного интеллекта. ОА-граф абстракций, содержащий в себе как описание данных, так и включающий в себя капсулы с миллипрограммой, представляет собой мощнейшую базу знаний. Свойство изоморфизма позволяет довольно легко изменять смысловой ОА-граф — это очень важное свойство для интеллектуальных систем, ведь в базу знаний будет необходимо постоянно добавлять новые сведения, что не должно приводить к нарушению целостности всей ВС. Весьма удобен и вывод смысловых конструкций от простого к сложному (конус абстракций) и от сложного к простому (обратный конус абстракций), что копирует мыслительный процесс человека.

Заключение

В настоящий момент реализована программная среда создания и запуска ОА-образа (рис. 3), что на практике подтверждает работоспособность предложенной архитектуры. В среде реализованы около 50 классов ВФУ, ОА-язык программирования (компилятор ОА-языка создан на базе ОА-архитектуры); имеются возможности не только программирования, но и управления ОА-системой в реальном времени, то есть коррекция алгоритма работы системы без перезагрузки ОА-программы; создания распределенных ОА-систем. В состав среды кроме ОА-компилятора входят инструменты, облегчающие работу с ОА-образом. С помощью разработанной среды решен ряд практических задач в области психофизиологии и робототехники.

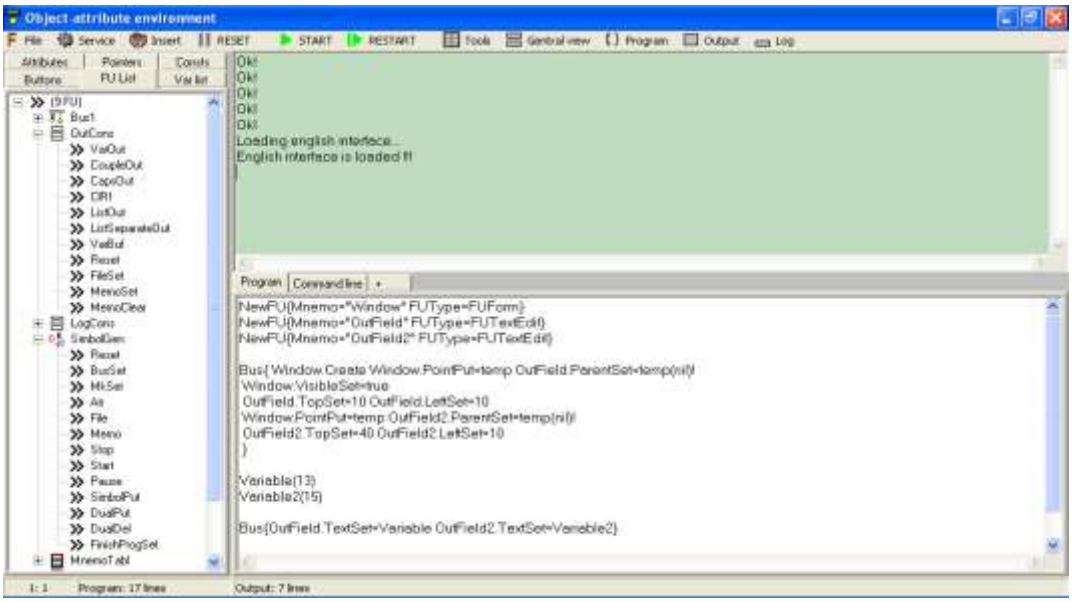

Рис. 3. Среда создания и запуска ОА-образа

Литература

1. Jurij Silk, Borut Robic and Theo Ungerer «Asynchrony in parallel computing: From dataflow to multithreading» Institut Jozef Stefan, Technical Report CDS-97-4, September 1997 Data flow computing: theory and practice / edited by John A. Sharp. Ablex Publishing Corp. Norwood, NJ, USA,1992

2. Jeck B. Dennis. Data Flow Supercomputers. Computer, November 1980

3. VEEN. Dataflow Machine Architecture. Center for Mathematics and Computer Science, P.O. Box 4079, 1009 AB Amsterdam, The Netherlands. 1987

4. А.П. Стасенко Обзор потоковых языков программирования. // Проблемы интеллектуализации и качества систем информатики. Сб. статей под редакцией доктора

физ.-мат. наук, профессора, чл.-корр. РАЕН В.Н. Касьянова. Серия "Конструирование и оптимизация программ". Новосибирск, 2006

5. Gabriel, R. 2002. Objects Have Failed: Notes for a Debate. (retrieved 17 May 2009). (http://www.dreamsongs.com/Files/ObjectsHaveFailed.pdf);

6. Joe Armstrong, Robert Virding, Claes Wikström, Mike Williams. Ericsson Concurrent Programming in ERLANG. Second Edition. Telecommunications Systems Laboratories, Box 1505, S - 125 25 Älvsjö, Sweden (http://www.erlang.se/publications/erlang-book-part1.pdf)

7. The Scala Language Specification Version 2.7 (http://www-edlab.cs.umass.edu/cs530/ScalaReference.pdf)

8. В.А. Торгашев, И.В. Царёв Семейство суперкомпьютеров с динамической архитектурой – концептуальные основы. Санкт-Петербургский институт информатики и автоматизации Российской академии наук, г. Санкт-Петербург, Россия (http://www.nbuv.gov.ua/portal/natural/ii/2009_3/5/00_Torgashev_Tsarev.pdf)

9. А. В. Мыскин, В. А. Торгашев, И. В. Царев. Процессоры с динамической архитектурой на основе схем гибкой логики // Труды СПИИРАН. Вып. 1, т. 1. — СПб: СПИИРАН, 2002. (http://www.mathnet.ru/links/25552801c1cd6ead203a2b341e53318f/trspy_75_card_rus.pdf)

10. Салибекян С.М. Принципы милликомандной архитектуры как основа построения высокопроизводительных адаптивных вычислительных систем // Автоматизация и современные технологии. 2002. № 5. – Стр. 25-32.

11. Салибекян С.М., Панфилов П.Б ОА-архитектура построения и моделирования распределенных систем автоматизации // Автоматизация в промышленности. N11, 2010. - Стр .51-56.

12. ОА-архитектура – новый подход к созданию объектных систем // Объектные системы - 2011: материалы III Международной научно-практической конференции (Ростов-на-Дону 10-12 мая 2011 г.) / Под общ. ред. П.П. Олейника. - Ростов-на-Дону, 2011. - С. 73-79 (http://objectsystems.ru/files/Object_Systems_2011_Proceedings.pdf)

Сведения об авторах:

Салибекян Сергей Михайлович – старший преподаватель кафедры «Вычислительные системы и сети», ФГБОУ ВПО «Московский государственный институт электроники и математики (технический университет)» (МИЭМ), г. Москва

В 1994 году окончил ГОУ ВПО «Московский государственный институт электроники и математики (технический университет)»

Более 15 печатных работ.

Область научных интересов: распределенные вычислительные системы, вычислительные системы, управляемые потоком данных (dataflow).

Контактный телефон: (495) 916-89-09

Адрес электронной почты: salibek@yandex.ru

Панфилов Петр Борисович – к.т.н., доцент, профессор кафедры «Вычислительные системы и сети», ФГБОУ ВПО «Московский государственный институт электроники и математики (технический университет)» (МИЭМ), г.Москва

В 1982 году окончил Московский институт электронного машиностроения (МИЭМ).

Кандидат технических наук, доцент.

Более 60 печатных работ, 1 монография.

Область научных интересов: системы реального времени, моделирование виртуальных сред, визуальный компьютинг, человеко-машинные системы

Контактный телефон: (495) 916-89-09

Адрес электронной почты: panfilov@miem.edu.ru