УДК 681.3

А.Е. Дорошенко, Е.А. Яценко

СРЕДСТВА СЕРВИСНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

Предложен подход к проектированию и генерации Грид-сервисов для платформы Globus Toolkit на основе использования разработанного алгеброалгоритмического интегрированного инструментария. Особенностью разрабатываемой авторами системы является применение высокоуровневых спецификаций сервисно-ориентированных программ, не зависящих от реализации на конкретном языке программирования, а также использование метода диалогового конструирования синтаксически правильных программ. Подход проиллюстрирован на примере разработки Грид-сервиса, осуществляющего параллельную распределенную сортировку массивов. Проведен эксперимент по выполнению разработанной сервисно-ориентированной программы на локальном кластере.

Введение

За последнее десятилетие значительно возрос интерес к Грид вычислениям как способу предоставления прикладным разработчикам возможности соединять разрозненные ресурсы для решения крупномасштабных научных проблем [1]. С возрастанием сложности и динамичности Грид приложений использование программных средств для их разработки становится существенным при настройке параметров приложений или обнаружении дефектов программы. Одной из наиболее популярных Грид платформ является сервисно-ориентированная платформа Globus Toolkit [2]. Globus обеспечивает программную инфраструктуру, которая дает возможность приложениям работать с распределенными разнородными вычислительными ресурсами как с единой виртуальной машиной. Для упомянутой платформы существует инструментарий Introduce [3], позволяющий упростить разработку и развертывание Грид-сервисов засчет сокрытия низкоуровневых деталей Globus Toolkit. Introduce предоставляет пользователю графический интерфейс для создания сервисов, а также добавления методов, ресурсов, свойств и ограничений безопасности. Разработчику затем необходимо реализовать логику методов, не беспокоясь об архитектурных аспектах Грид-сервиса, генерации кода клиента или сервера, а также синхронизации всех необходимых файлов. Однако, написание кода методов по прежнему является достаточно сложной задачей. Целью данной работы является автоматизация разработки алгоритмов, представляющих логику выполнения сервисов и программ-клиентов, а также генерация соответствующего кода в языке программирования. Данная автоматизация осуществляется с помощью разработанного Интегрированного инструментария Проектирования и Синтеза программ (“ИПС”) [4]. “ИПС” предназначен для интерактивного конструирования алгоритмов с использованием высокоуровневых алгебраических спецификаций (систем алгоритмических алгебр Глушкова) и генерации программ на целевом языке программирования (C, C++, Java). В предыдущих работах упомянутый инструментарий был использован для разработки многопоточных программ с общей памятью [5] и параллельных программ, ориентированных на передачу сообщений (MPI) [6]. В данной работе была произведена настройка “ИПС” на разработку сервисно-ориентированных программ для Грид.

Материал работы состоит из таких разделов. В разделе 1 охарактеризован Globus Toolkit и система автоматизации разработки Грид-сервисов Introduce. В разделе 2 рассматривается разработанный интегрированный инструментарий, применяемый для проектирования и генерации сервисов, а также используемые совместно с инструментарием онтологические средства. В разделе 3 приведен пример Грид-сервиса, разработанного с использованием Introduce и “ИПС”.

1. Globus Toolkit и Introduce

Вычислительная сеть, построенная на основе Globus является программно-аппаратной инфраструктурой, которая обеспечивает надежный, непротиворечивый и проникающий доступ к предельным вычислительным возможностям систем, несмотря на географическое размещение ресурсов и их потребителей [7]. Центральным элементом системы Globus является инструментарий Globus Toolkit, который определяет основные сервисы и возможности, требующиеся для создания Grid. Инструментарий состоит из набора компонентов, реализующих базовые сервисы, такие, как защита, размещение ресурса, управление ресурсами и связь.

Инструментарий Globus Toolkit обеспечивает набор сервисов, которые разработчики инструментальных средств или приложений могут использовать для их специфических нужд. Он создан как слоистая архитектура, в которой сервисы верхнего глобального уровня смонтированы на уже существующих низкоуровневых локальных сервисах ядра. Инструментарий является модульным и приложение может использовать функции Globus без библиотеки обменов. Инструментарий Globus состоит из следующих функций (точный набор зависит от версии):

• основанный на протоколе HTTP администратор распределения ресурсов GRAM − используется для распределения вычислительных ресурсов, контроля и управления вычислением на этих ресурсах;

• расширенная версия протокола передачи файлов GridFTP − применяется для доступа к данным; расширения включают использование протоколов защиты уровня связности, частичного доступа к файлу и управления параллелизмом для высокоскоростных передач;

• аутентификация и связанные с ней сервисы безопасности GSI (Grid Security Infrastructure);

• распределенный доступ к информации о структуре и состоянии, который основан на протоколе облегченного доступа к каталогам LDAP (Lightweight Directory Access Protocol); этот сервис используется для определения стандартного протокола информации о ресурсе и связанной с ним информационной модели;

• удаленный доступ к данным через последовательный и параллельный интерфейсы GASS (Global Access to Secondary Storage), включая интерфейс к GridFTP;

• построение, кэширование и размещение выполнимых программ GEM (Globus Executable Management);

• резервирование и распределение ресурсов GARA (Globus Advanced Reservation and Allocation).

Инструментарий Introduce [3] предназначен для поддержки трех основных этапов разработки Грид-сервисов для Globus:

1) создание базисной структуры сервиса. Разработчик сервиса описывает на верхнем уровне основные свойства сервиса, такие как имя и пространство имен. После того как пользователь указал упомянутые основные свойства, Introduce создает базисную реализацию сервиса, к которой пользователь может затем добавлять методы и опции безопасности на этапе изменения сервиса;

2) модификация сервиса. Этап модификации позволяет разработчику добавлять, удалять и изменять методы и свойства сервиса, ресурсы, контексты сервиса, и безопасность на уровне сервиса или методов. На этом шаге разработчик может создать строго-типизированный интерфейс сервиса, определяя входные и выходные параметры методов сервиса. После того как операции были добавлены к сервису, разработчик может добавлять логику, реализующую методы;

3) развертывание. Разработчик может развернуть сервис, который был создан с помощью Introduce в контейнер Грид сервисов (например, контейнер сервисов Globus или Tomcat).

Для выполнения перечисленных трех шагов разработчик сервиса использует графическую среду Introduce (рис. 1). По описанию сервиса Introduce генерирует каркас сервиса и соответствующей программы-клиента, которые разработчику необходимо заполнить кодом реализации.

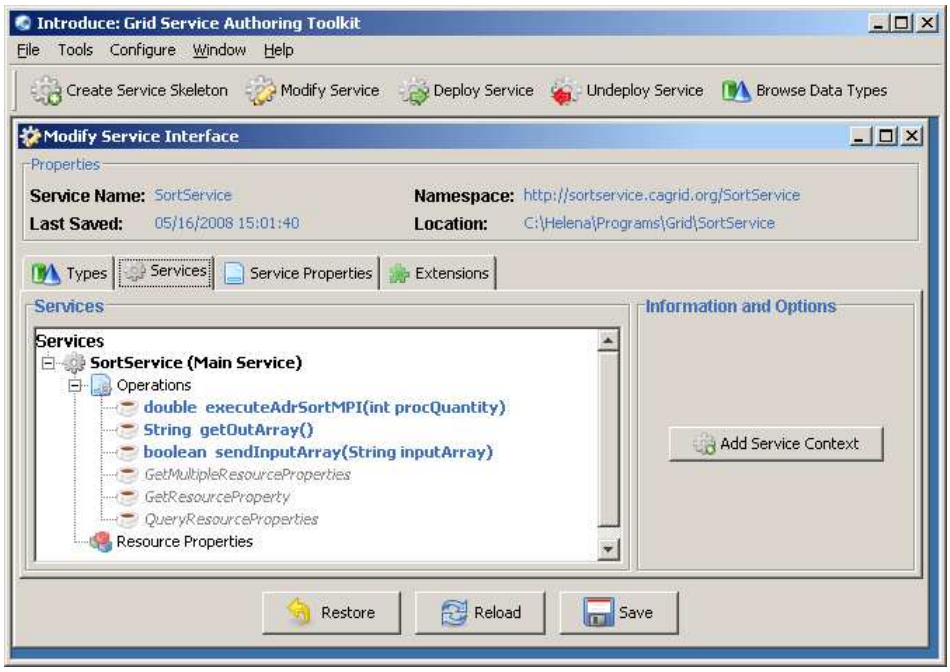

Рис. 1. Окно системы Introduce с описанием функций Грид-сервиса

2. Инструментарий “ИПС” и онтологические средства проектирования программ

В работах [4–6] разработан инструментарий “ИПС”, позволяющий автоматизировать проектирование и генерацию программ по формализованным спецификациям в САА. В данной работе он был применен для конструирования сервисно-ориентированных программ. А именно, он используется для конструирования алгоритмов методов Грид-сервисов и соответствующих программ-клиентов, а также генерации соответствующего текста на языке программирования с заполнением каркасного кода, созданного с помощью программы Introduce (см. раздел 1).

Основным компонентом инструментария является ДСП-конструктор (рис. 2) – диалоговый конструктор синтаксически правильных программ. Он позволяет проектировать алгоритмы в диалоговом режиме, выбирая из списка заранее заготовленные конструкции (операции САА, базисные операторы и условия). “ИПС” использует различные формы представления алгоритмов в процессе их проектирования – естественно-лингвистическую (САА-схемы), алгебраическую (регулярные схемы) и граф-схемную. Процесс проектирования алгоритмов является по уровневым (осуществляется сверху-вниз) и представлен в виде дерева конструирования (рис. 2). Выбираемые пользователем конструкции, а также переменные, входящие в них, отображаются в дереве с дальнейшей детализацией переменных. В зависимости от типа выбранной переменной (операторной или логической) система предлагает соответствующий список операций САА или базисных понятий. Дерево для каждой из функций схемы алгоритма (методов сервиса) приводится на отдельной вкладке окна “Дерево алгоритма”. Текст САА-схемы, соответствующий дереву конструирования, отображается в отдельном текстовом окне.

По полученной в процессе конструирования схеме, а также реализациями элементарных операторов и условий на целевом объектно-ориентированном языке программирования, которые хранятся в БД “ИПС”, ДСП конструктор выполняет синтез программы. На вход синтезатора поступает также файл, который содержит каркасное описание основного класса про-

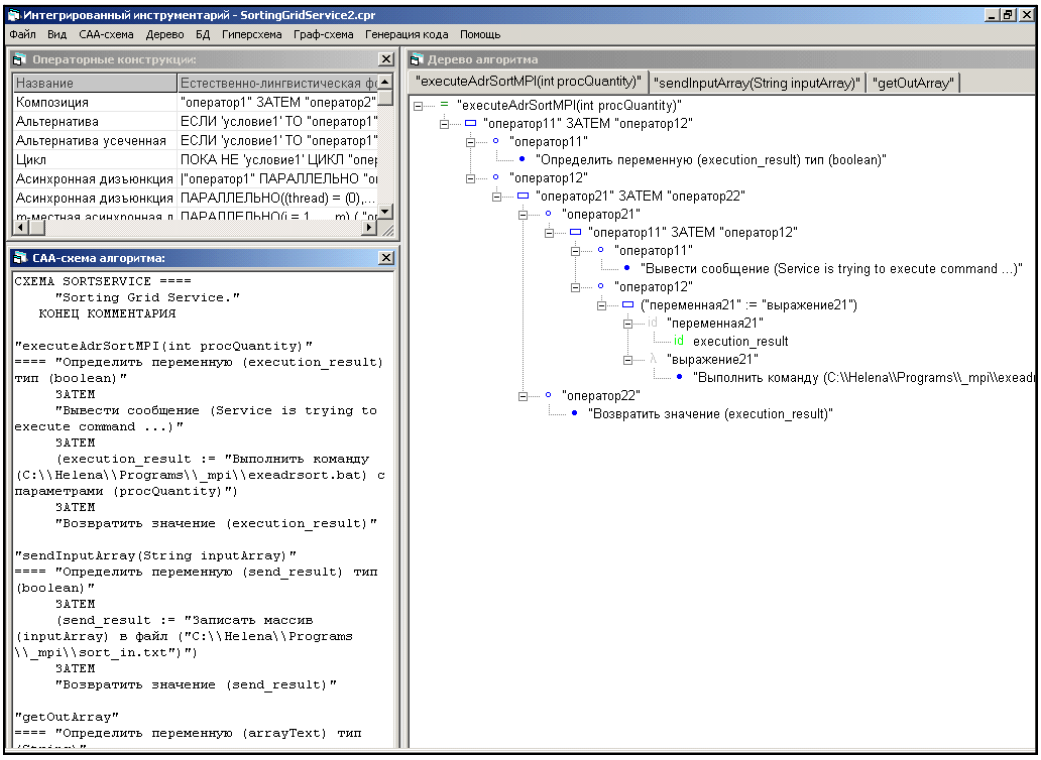

Рис. 2. Окно ДСП-конструктора с примером сервиса

граммы (без реализаций методов), в который выполняется подстановка кода. Такой файл может быть, например, сгенерирован на основе диаграммы классов в системе Rational Rose [8]. В случае сервисно-ориентированной программы для Globus Toolkit для генерации каркасного кода используется инструментарий Introduce. Способ подстановки сгенерированного кода для методов (функций) в каркасный файл указывается в параметрах генерации.

В работе [9] для проектирования программ совместно с интегрированным инструментарием был использован аппарат онтологий. В рамках разработанной методологии с помощью онтологии предметной области описывается каркас разрабатываемого алгоритма – обрабатываемые данные; названия функций, их входные и выходные параметры, а также взаимосвязи между функциями. В качестве средства разработки онтологии была выбрана система Protégé. Созданная в Protégé онтология передается в инструментарий “ИПС”, в котором осуществляется генерация соответствующей каркасной схемы алгоритма, а также конструирование реализаций функций алгоритма. Упомянутые средства были использованы для проектирования MPI программы сортировки.

В данной работе приведенная в [9] онтология была дополнена новым классом GRID_PROGRAM и двумя его подклассами – GridService и GridServiceClient. На рис. 3 приведен сокращенный вид модифицированной онтологии. Класс Data представляет различные структуры данных, используемые в Грид программах. Класс Operation отображает операции, применяемые к данным в программе – операторы и предикаты. Операции могут быть базисными или составными. Онтология также содержит класс PROGRAM для описания произвольных программ, которые могут использоваться Грид-сервисами, например, MPI программ. Свойства классов Data, Operation и PROGRAM, а также метод генерации каркасного алгоритма по онтологии подробно рассмотрены в [9].

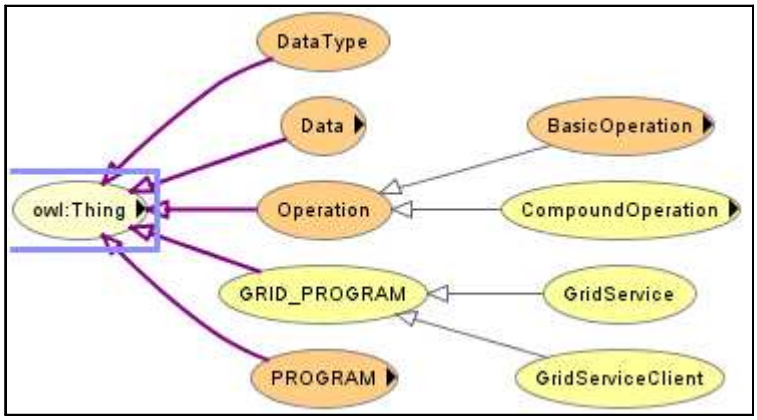

Рис. 3. Онтология, описывающая концепции для проектирования Грид программ

Свойствами класса GridService являются его название, комментарий, используемые ресурсы и совокупность методов:

$$HasName : GridService \xrightarrow{1:1} string$$

$$Comment : GridService \xrightarrow{1:1} string$$

$$HasResource : GridService \xrightarrow{1:n} Data$$

$$HasMethod : GridService \xrightarrow{1:n} CompoundOperation$$

Свойствами класса GridServiceClient являются следующие:

$$HasName : GridServiceClient \xrightarrow{1:1} string$$

$$Comment : GridServiceClient \xrightarrow{1:1} string$$

$$IsClientOf : GridServiceClient \xrightarrow{1:1} GridService$$

$$UsesData : GridServiceClient \xrightarrow{1:n} Data$$

$$HasMethod : GridServiceClient \xrightarrow{1:n} CompoundOperation$$

3. Пример сервиса, разработанного с использованием “ИПС”

Рассмотрим Грид-сервис, предназначенный для сортировки массивов, а также соответствующую ему программу-клиент, спроектированные с использованием “ИПС”. Грид-сервис SortService выполняет параллельную сортировку переданного ему программой-клиентом численного массива. Для упорядочения массива сервис вызывает параллельную MPI программу адресной сортировки. САА-схема данной сортировки разработана и приведена в работе [9]. По окончании сортировки сервис возвращает упорядоченный массив клиенту.

Рассмотрим структуру Грид-сервиса. Он включает в себя три метода:

- sendInputArray(String inputArray) – функция, выполняющая запись переданного ей массива inputArray в файл для последующей сортировки MPI программой; массив передается в виде текстовой строки (String). Метод возвращает истинное значение, если запись в файл была произведена успешно, и ложное в противном случае;
- executeAdrSortMPI(int procQuantity) – функция, выполняющая запуск MPI программы адресной сортировки с количеством процессов procQuantity. Запуск производится на той же машине, где распо-

ложен Грид-сервис. Метод возвращает истинное значение, если MPI программа выполнилась успешно, и ложное в противном случае;

• getOutArray() – функция, возвращающая отсортированный массив в виде текстовой строки. Массив считывается из файла, записанного MPI программой сортировки.

Каркас программного кода на языке Java для Грид-сервиса SortService был сгенерирован с помощью программы Introduce. Дальнейшее проектирование алгоритмов методов (в языке САА) и генерация соответствующего кода выполнена в инструментарии “ИПС”.

В процессе проектирования Грид-сервиса совместно с “ИПС” также были использованы онтологические средства, описанные в разделе 1. В табл. 1 приведены значения свойств для экземпляра Грид-сервиса SortService, определенного в онтологии. В табл. 2 приведен пример онтологического описания метода sendInputArray. Аналогичным образом определяются также методы executeAdrSortMPI и getOutArray.

По онтологическому описанию рассматриваемого Грид-сервиса и его методов в инструментарии “ИПС” была сгенерирована следующая каркасная САА-схема:

```
СХЕМА SORTSERVICE ====
     "Sorting Grid Service"
   КОНЕЦ КОММЕНТАРИЯ

"sendInputArray
(String inputArray)"
==== "Записать массив (array) в
     файл (file)"

"executeAdrSortMPI
(int procQuantity)"

==== "Выполнить команду
     (executable_file) с
     параметрами
     (parameters_string)"

"getOutArray()"
==== "Прочитать массив (array) из
     файла (file)"
КОНЕЦ СХЕМЫ SORTSERVICE
```

Каждый метод в приведенной схеме содержит вызов оператора в соответствии с его онтологическим описанием. Например, схема оператора sendInputArray содержит вызов базисного оператора записи массива в файл в соответствии со значением свойства UsesOperation, приведенным в табл. 2. Далее в ДСП-конструкторе необходимо продолжить проектирование Грид-сервиса на основе сгенерированной каркасной схемы. В данном случае необходимо установить параметры базисных элементов и сконструировать алгоритмы функционирования методов, используя присутствующие в схеме вызовы операций, а также выбирая другие операции из списка в ДСП-конструкторе.

Далее представлена полная детализация САА-схемы Грид-сервиса, построенная в ДСП-конструкторе интегрированного инструментария (рис. 2). В схеме названия методов с описанием формальных параметров отделены от реализации методов цепочками символов “====”. Текст базисных и составных операторов взят в двойные кавычки.

```
СХЕМА SORTSERVICE ====
     "Sorting Grid Service"
   КОНЕЦ КОММЕНТАРИЯ

"sendInputArray(String inpu-
tArray)"

==== "Определить переменную
     (send_result) тип (boolean)"

     ЗАТЕМ
     (send_result := "Записать
     массив (inputArray) в файл
     ("C:\\Programs\\mpi\\
     sort_in.txt")")

     ЗАТЕМ
     "Возвратить значение
     (send_result)"

"executeAdrSortMPI(int proc-
Quantity)"

==== "Определить переменную
     (execution_result) тип
     (boolean)"
     ЗАТЕМ
     (execution_result :=
```

Таблица 1. Значения онтологических свойств Грид-сервиса SortService

Свойство	Значение
HasName	SortService
Comment	Sorting Grid Service
HasResource	–
HasMethod	• sendInputArray • executeAdrSortMPI • getOutArray

Таблица 2. Значения онтологических свойств метода sendInputArray

Свойство	Значение
HasName	sendInputArray
Comment	Receives array from client
HasParameter	inputArray
OperationOutput	boolean
UsesOperation	Write_array_to_file

```
      "Выполнить команду
      (C:\\Programs\\mpi\\
      exeadrsort.bat) с
      параметрами (procQuantity)")
      ЗАТЕМ
      "Возвратить значение
      (execution_result)"
     "getOutArray()"
==== "Определить переменную
      (arrayText) тип (String)"
      ЗАТЕМ
      "Прочитать массив
      (arrayText) из файла
      ("C:\\Programs\\mpi\\
      sort_out.txt")"
      ЗАТЕМ
      "Возвратить значение
      (arrayText)"
      КОНЕЦ СХЕМЫ SORTSERVICE
```

В методе executeAdrSortMPI для запуска MPI программы сортировки используется базисный оператор

```
"Выполнить команду
(C:\\Programs\\mpi\\
exeadrsort.bat) с параметрами
(procQuantity)"
```

Таким образом, запуск программы сортировки производится из командного файла (exeadrsort.bat), содержащего вызов команды mpirun [10] с двумя параметрами – количеством параллельных процессов и названием исполняемого файла (MPI программы). Отметим, что для реализации на языке Java вышеприведенного базисного оператора использован метод exec класса Runtime [11], который создает отдельный процесс для выполнения команды (на локальной машине). Для ожидания завершения программы использован метод waitFor класса Process [11].

Рассмотрим далее сокращенную схему программы-клиента для приведенного Грид-сервиса.

```
СХЕМА SORTSERVICECLIENT ====
     "Client of Grid service
SortService"
   КОНЕЦ КОММЕНТАРИЯ
"GlobalData"
==== "Определить переменную
     (client) типа
     (SortServiceClient)";
     "Определить переменную
     (inputFile) типа (String)";
     "Определить переменную
     (numberOfProcesses) типа
     (int)";
     "Определить переменную
     (outputFile) типа (String)"
     "runProgram"
==== ЛОКАЛЬНЫЕ_ПЕРЕМЕННЫЕ
     (
       "Определить переменную
       (res) типа (boolean)";
```

```
  "Определить переменную
  (arrayText) типа
  (String)"
)
"Прочитать массив
(arrayText) из
файла (inputFile)"
ЗАТЕМ
ЕСЛИ НЕ('Значение
        (arrayText) пусто')
ТО (res :=
"sendInputArray(arrayText)")
ЗАТЕМ
ЕСЛИ 'Значение (res)
     истинно'
ТО
  (res :=
  "executeAdrSortMPI
   (numberOfProcesses)")
   ЗАТЕМ
   ЕСЛИ 'Значение (res)
        истинно'
   ТО
     (arrayText :=
     "getOutArray")
     ЗАТЕМ
     ЕСЛИ НЕ('Значение
     (arrayText) пусто')
     ТО
       (res :=
       "Записать массив
       (arrayText) в файл
       (outputFile)")
     КОНЕЦ ЕСЛИ
   КОНЕЦ ЕСЛИ
КОНЕЦ ЕСЛИ
КОНЕЦ ЕСЛИ
КОНЕЦ СХЕМЫ ORTSERVICECLIENT
```

Входными параметрами программы SortServiceClient являются имя входного файла (содержащего массив, который необходимо отсортировать), количество параллельных процессов и имя выходного файла. Взаимодействие клиента с сервисом отображено в функции runProgram() и является следующим. Вначале программа-клиент считывает массив из входного файла и пересылает массив сервису с помощью метода sendInputArray (описание данного метода приведено выше). Затем SortServiceClient вызывает функцию executeAdrSortMPI, в результате чего сервис запускает MPI программу сортировки. По окончании выполнения MPI программы клиент вызывает метод getOutArray для получения отсортированного массива, который затем записывает в выходной файл.

Грид-сервис SortService был выполнен на локальном кластере, состоящем из двух компьютеров:

- AMD Athlon 64 Processor 3000+, 1.81 GHz, 1,00 GB RAM;
- Intel Pentium D CPU 3.4 GHz, 1,00 GB RAM (2-ядерный).

Результаты выполнения приведены на рис. 4 и в табл. 3, и показывают хорошую степень распараллеливаемости вычислений. При запуске трех процессов учитывался тот факт, что процессор Intel Pentium D является 2-ядерным.

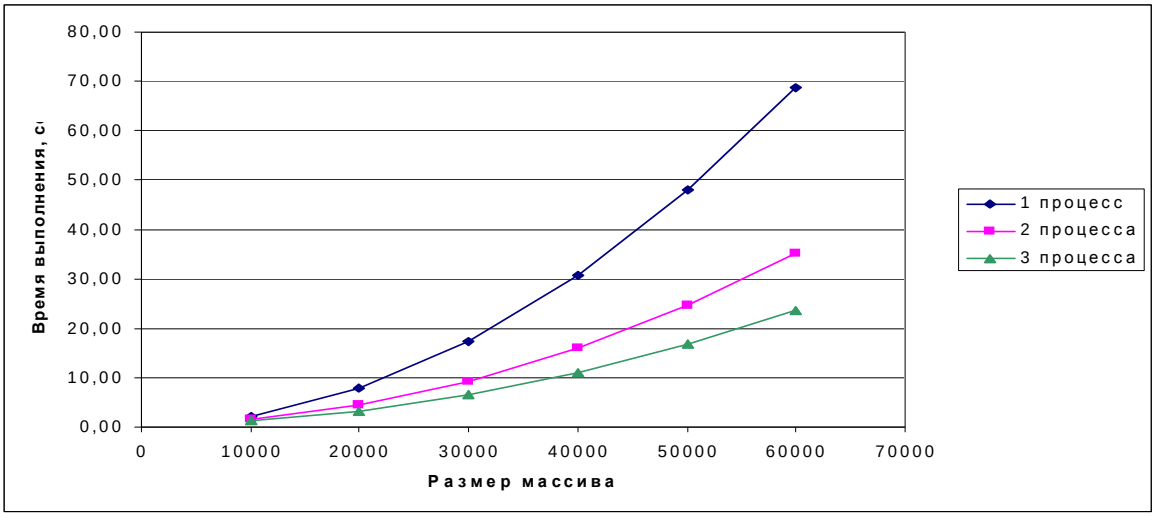

Рис. 4. Зависимость времени выполнения функции executeAdrSortMPI Грид-сервиса SortService от размера входного массива

Таблица 3. Время выполнения функции executeAdrSortMPI Грид-сервиса SortService, которая вызывает MPI программу адресной сортировки

Размер массива	Время t_n выполнения функции при использовании n процессов, сек			Ускорение, t_1/t_2	Ускорение, t_1/t_3
	t_1	t_2	t_3		
10000	1,98	1,52	1,23	1,31	1,61
20000	7,77	4,43	3,21	1,76	2,42
30000	17,39	9,22	6,46	1,89	2,69
40000	30,69	15,98	10,92	1,92	2,81
50000	48,13	24,72	16,80	1,95	2,87
60000	68,61	35,04	23,66	1,96	2,90

Заключение

В работе предложен подход к проектированию и генерации Грид-сервисов для платформы Globus Toolkit на основе использования системы Introduce, алгебро-алгоритмического инструментария "ИПС", а также онтологических средств. Преимуществом разрабатываемой авторами системы "ИПС" является применение высокоуровневых спецификаций сервисно-ориентированных программ, не зависящих от реализации на конкретном языке программирования, а также использование метода диалогового конструирования синтаксически правильных программ. Подход апробирован на разработке Грид-сервиса, осуществляющего параллельную распределенную сортировку массивов (на основе вызова MPI программы). Проведен эксперимент по выполнению разработанной сервисно-ориентированной программы на локальном кластере, который показал хорошую степень распараллеливаемости и масштабируемости вычислений.

К перспективам дальнейшего развития разрабатываемых инструментальных средств является их применение для разработки сервисов, выполняющих вычисления на гетерогенном кластере.

1. *Prodan R., Fahringer T.* Grid Computing Experiment Management, Tool Integration, and ScientificWorkflows. – Springer-Verlag, Berlin, Heidelberg, 2007. – 317 p.
2. *Globus* Toolkit Site. – http://www. globus.org
3. *Introduce* Main Project Site. – http://www.cagrid.org/mwiki/index.php?title=Introduce.
4. *Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А.* Алгеброалгоритмические модели и методы параллельного программирования. – Киев: Академпериодика, 2007. – 631 с.
5. *Дорошенко А.Е., Жереб К.А., Яценко Е.А.* Об оценке сложности и координации вычислений в многопоточных программах // Проблемы программирования. – 2007. – № 2. – С. 41–55.
6. *Дорошенко А.Е., Жереб К.А., Яценко Е.А.* Средства синтеза параллельных MPI программ // Проблемы программирования. – 2008. – № 2–3. – С. 595–604.
7. *Дорошенко А.Е., Алистратов О.В., Тырчак Ю.М. и др.* Системы Grid-вычислений – перспектива для научных исследований // Проблемы программирования. – 2005. – № 1. – С. 14–38.
8. *Квапрани Т.* Rational Rose 2000 и UML. Визуальное моделирование: Пер. с англ. – М.: ДМК, 2001. – 176 с.
9. *Дорошенко А.Е., Яценко Е.А.* Средства автоматизации разработки параллельных программ на основе онтологий и алгебр алгоритмов // Проблемы программирования. – 2008. – № 4. – С. 94–103.
10. *MPICH* site. – www.mcs.anl.gov/mpi/mpich/
11. *Холл М., Браун Л.* Программирование для Web. Библиотека профессионала: Пер. с англ. – М.: Вильямс, 2002. – 1264 с.

Получено 16.03.2009

Об авторах:

Дорошенко Анатолий Ефимович,
доктор физико-математических наук,
профессор, заведующий отделом теории
компьютерных вычислений,

Яценко Елена Анатольевна,
кандидат физико-математических наук,
старший научный сотрудник.

Место работы авторов:

Институт программных систем
НАН Украины.
Тел.: (044) 526 1538,
e-mail: dor@isofts.kiev.ua, aiyat@i.com.ua