

Экспериментальный анализ эффективности использования языка Ада95 в распределенных вычислительных системах

**Корочкин А., Ровто В.,
Корочкин Д.**

Национальный технический университет Украины – Киевский политехнический институт

Вступление. Широкое распространение распределенных вычислительных систем (РВС) для решения сложных вычислительных задач связано с проблемой разработки эффективного программного обеспечения для таких систем, которая должна обеспечиваться соответствующими языковыми или библиотечными средствами. В настоящее время программирование для РВС осуществляется с использованием разнообразных средств:

- сокетов
- удаленных подпрограмм
- параллельной виртуальной машины (PVM)
- интерфейса посылки сообщений (MPI)
- CORBA, COM/DCOM.

которые поддерживают различные уровни доступа к аппаратуре и соответственно различные виды удобства разработки распределенных приложений и последующую эффективность его выполнения [8, 9, 12].

В настоящей работе рассматривается возможность использование языка Ада95 для программирования в РВС. Ада95 - второй стандарт известного языка программирования Ада. Со времени своего создания (1986 год) язык широко использовался во многих известных проектах [1]. В настоящее время ведутся работы по созданию очередного стандарта языка (Ада9х), в котором будут учтены и реализованы все новейшие требования к современным языкам программирования

Наличие в существующем стандарте языка приложения Annex E - "Distributed Systems" свидетельствует о том, что в нем вопросам программирования для РВС уделено особое внимание. В стандарте определены понятия распределенной вычислительной системы, распределенной программы, механизмов взаимодействия, средств поддержки и конфигурирования [2].

Учитывая, что современные высокопроизводительные РВС (кластерные системы) строятся, как правило, на основе объединения многопроцессорных систем, в частности СМП систем, актуальным при разработке распределенных приложений остаются вопросы эффективной организации в программе параллельных вычислений. С этой точки зрения язык Ада95 известен своими мощными механизмами работы с параллельными процессами.

Таким образом, язык Ада95 потенциально обеспечивает мощную поддержку разработки программного обеспечения для РВС и может быть использован для программирования параллельных и распределенных приложений.

Цель работы. Целью работы является повышение эффективности организации вычислительных процессов в РВС, базирующееся на использовании языка программирования Ада95. С этой целью в работе осуществлен анализ возможностей языка для разработки распределенных приложений, выполнена разработка с использованием языка Ада95 пакета прикладных программ и проведены экспериментальные исследования их эффективности в реальной РВС.

Структура РВС. Структура РВС, используемой для проведения экспериментальных исследований, приведена на рисунке 1. РВС включает 8 узлов, оснащенных двухпроцессорными СМП системами, и 8-и портовый коммутатор. Сетевая организации РВС базируется на использовании 100 Mb Ethernet.

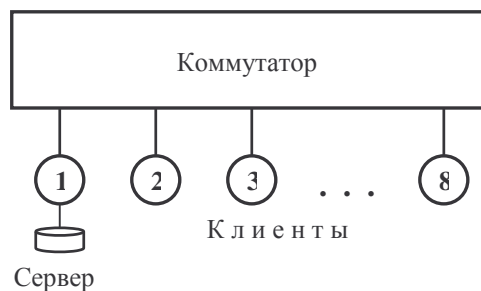


Рис 1. Структура РВС

Рассматриваемая структурная организация РВС включает 16 процессоров Xeon и обеспечивает параллельную обработку на двух уровнях:

- внутри каждого узла за счет использования в нем двухпроцессорной СМП системы
- между узлами РВС.

Разработка приложения для рассматриваемой РВС связана с решением следующих задач:

- разработкой параллельного (распределенного) алгоритма решения исходной вычислительной задачи
- разработки алгоритмов клиентской и серверной частей приложения
- формирования распределенный частей (разделов) программы и схемы их взаимодействия
- формирование для каждого узла набора параллельных взаимодействующих процессов и решения для них задач взаимного исключения и синхронизации

Для решения указанных проблем необходимы средства, обеспечивающие эффективное программирование параллельных и распределенных

процессов. Рассмотрим и проанализируем детально возможности языка Ада95 с этой точки зрения.

Параллельная обработка. Концепция процессов в языке базируется на модели взаимодействующих последовательных процессов Хоара [11]. Организация параллельных процессов в Аде основывается на использовании специального вида модулей – задач (*task*). Такой модуль имеет спецификацию, которая определяет его интерфейс для взаимодействия с другими модулями и тело, определяющее его действия при выполнении. Задачный тип (*task type*) позволяет создавать группы задач. Используемый для параметризации задачного типа дискриминант позволяет формировать идентификатор задачи (*Tid*) и вносить коррективы в поведение каждой задачи группы. Это особенно ценно при программировании для масштабируемых РВС, в которых число процессов может меняться в зависимости от числа используемых узлов (процессов).

Ада реализует трехэтапный процесс выполнения задачи: создание задачи, ее активизация и выполнение тела задачи. Не имея явных средств запуска задачи на выполнение, (задача, описанная в программе, автоматически стартует при запуске этой программы), язык позволяет реализовать и варианты явного запуска задачи с помощью дополнительных процедур, где выполняется описание вложенных в них задач, а также через ссылочные типы.

Корректное поведение параллельной программы зависит от реализации взаимодействия процессов. При использовании СМП систем в узлах РВС такое взаимодействие основывается на использовании общих (разделяемых) переменных. В этом случае взаимодействие процессов связано с решением задач взаимного исключения и синхронизации.

Задачи взаимного исключения и синхронизации процессов. Для решения задачи взаимного исключения Ада95 предоставляет набор средств в виде *atomic* операций, семафоров, защищенных типов/объектов [1, 2, 6].

Atomic операции обеспечивают синхронизированный доступ к разделяемым ресурсам (переменным и типам) с помощью прагм *Atomic* и *Volatile*.

Механизм семафоров в языке реализован в виде пакета *Synchronous_Task_Control*. Семафорный тип *Suspension_Object* позволяет использовать двоичные (логические) семафоры, для которых в пакете определены операции в виде процедур *Suspend_Until_True()*, *Set_True()*, *Set_False()*.

Механизм мониторов обеспечивает высокоуровневый средства работы с процессами и в языке представлен защищенными модулями (*protected objects/types*). Ада95 реализует продвинутую модель монитора, основанную на условных мониторах Хоара. Три вида защищенных операций (подпрограммы, функции и входы) обеспечивают несколько видов синхронизированного доступа к общему ресурсу, в частности функции

поддерживают параллельное считывание, а входы – доступ по условию [5].

Для синхронизации процессов можно использовать семафоры и входы в защищенных модулях. Наибольший интерес представляет использование защищенных входов. Наличие барьера в описании входа позволяет программировать различные виды условной синхронизации и объединять синхронизацию с взаимным исключением.

Распределенная обработка. Приложение Annex E определяет особенности структуры РВС и процесса разработки распределенного Ада приложения. РВС – набор узлов, которые разделяются на процессорные узлы (*processing nodes*) и узлы памяти (*storage nodes*) и предназначенных, соответственно, для обработки и хранения информации. Распределенная программа – набор активных и пассивных разделов (*partitions*), которые могут взаимодействовать между собой на основе удаленного доступа. Процесс конфигурирование разделов предполагает распределение их по узлам РВС.

Основным инструментом при построении распределенного приложения является набор специальных прагм категорий (*categorization pragmas*):

- Pure
- Shared Passive
- Remote Types
- Remote Call Interface.

Прагмы категории предназначены для управления глобальными данными, разделяемыми разделами; объявления типов, используемых для коммуникации разделов; описания интерфейса вызова удаленных процедур. Прагма *Asynchronous* позволяет организовать асинхронный вызов удаленной процедуры, при котором вызывающий раздел не ждет завершения удаленной процедуры в вызванном разделе. Пример спецификации пакета с использованием рассмотренных прагм, обеспечивающий удаленные вызовы процедур *Счет()* и *Поиск()*:

```
package Удаленные_Ресурсы is
  pragma Remote_Procedure_Interface;
  procedure Счет (X : in Data;
                 Y : out Data);
  procedure Поиск(X : in Элемент);
  pragma Asynchronous(Поиск);
end Удаленные_Ресурсы;
```

Annex E предоставляет пользователю важный механизм разработки программ для РВС - подсистему коммуникации разделов в виде пакета *System.RPC*. Пакет содержит набор типов и подпрограмм, обеспечивающих организацию коммуникации между активными разделами распределенного приложения. В частности, тип *Partition_ID* можно использовать для идентификации разделов, а процедуры *DO_RPC* и *DO_APC* для отправки сообщений между активными разделами, описанных с помощью типа *Partition_ID*.

Важной составляющей процесса разработки любого распределенного приложения является формирование разделов и их распределение по узлам (конфигурирование). В стандарте языка не определены средства для конфигурирования, которые должны быть реализованы непосредственно при реализации приложения Annex E. В работе использовалось программное обеспечение GNAT, в состав которого входит подсистема GLADE, включающая средства конфигурирования. Для формирования разделов и их привязки к узлам PBC, GLADE требует описания файла конфигурации с помощью специального языка конфигурирования, имеющего Ада - подобный синтаксис. Язык позволяет описать разделы распределенной программы, связать их с соответствующими библиотечными модулями, указать узел и место размещения (директорию) непосредственно в узле PBC, определить форму запуска приложения, возможность сжатия (архивирования) передаваемых данных и др. Ниже приведен пример файла конфигурации Конф_Файл.cfg в системе GLADE:

```
configuration Конф_Файл is
  Раздел_1 : Partition := ();
  Procedure Main is in Раздел_1;
  Раздел_2 : Partition :=
    (Удаленные_Ресурсы);
  for Раздел_2'Host use "host22";
  for Раздел_1'Storage_Dir use
    "/ds/bin;
  Канал_12 : Channel := (Раздел_1,
    Раздел_2);
  for Канал_12 use "ZIP";
end Конф_Файл;
```

Файл конфигурации в GLADE должен быть обработан программой gnatdist. GLADE также выполняет формирование заглушек (stub), необходимых для коммуникации разделов, размещенных в разных узлах PBC.

Экспериментальные исследования. Проведение экспериментальных исследований было связано с определением эффективности использования языка Ада95 в реальной PBC. С этой целью был разработан пакет программ для решения в PBC ряда задач линейной алгебры, включающий операции над векторами и матрицами большой размерности, решение систем линейных алгебраических уравнений, задач линейного программирования и математической физики. Для каждой задачи определялось T_p - время выполнения в PBC с изменяемым количеством (P) узлов. Используя значение T_0 - время выполнения задачи в однопроцессорном компьютере, определялся коэффициент ускорения $K_u = T_0/T_p$.

При программировании взаимодействия процессов в узлах PBC использовались различные механизмы (atomic операции, семафоры, защищенные модули и их комбинации), для организации взаимодействия узлов использовался механизм вызова удаленных методов (RPC - механизм) и сокеты [4,10].

В качестве альтернативы языку Ада95 была рассмотрена возможность использования языком

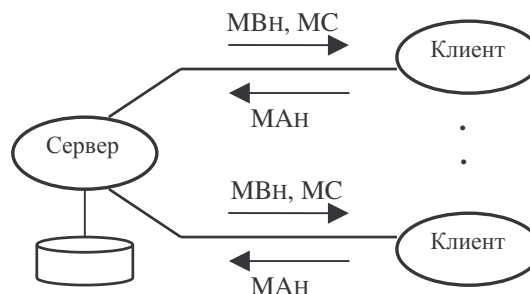
Java для программирования в PBC. Язык Java обладает развитыми средствами работы с процессами в виде потоков (threads), синхронизированными методами и блоками для взаимного исключения, методами wait()/notify() для синхронизации процессов [12]. Сравнительный анализ средств параллельной обработки языков Ада и Java выполнен в [7]. Язык Java позволяет программировать распределенные приложения с помощью сокетов и удаленных методов (RMI - механизм) [4].

В работе для рассмотренных математических задач также был разработан пакет программ на языке Java для реализации в PBC предложенной структуры, для которых были проведены аналогичные экспериментальные исследования. Это позволило сравнить как эффективность различных средств одного языка, так и возможности языков Ада и Java.

В качестве примера, иллюстрирующего методологию исследований, рассмотрим задачу умножения матриц $MA = MB * MC$ размерности $N*N$, лежащей в основе многих вычислительных задач и характеризующейся значительным объемом вычислений ($N*N*N$ операций) для больших N.

Используемая для задачи $MA = MB * MC$ модель клиент-сервер ориентирована на использование одного сервера и P клиентов. Параллельный математический алгоритм решения задачи умножения матриц представлен в виде $MA_n = MB_n * MC$, где MB_n - N строк (столбцов) матрицы. То есть каждый узел PBC формирует часть MA_n результирующей матрицы MA. Значение N определяется как $N = N/P$.

Сервер вводит значения исходных матриц MB и MC, рассылает полную матрицу MC и часть MB_n по узлам PBC, затем собирает от узлов результаты MA_n , формирует и выводит окончательный результат - матрицу MA. Для этого в сервере создается P процессов, каждый из которых выполняет взаимодействие с соответствующим узлом PBC. Каждый клиент получает исходные данные от сервера, выполняет счет $MA_n = MB_n * MC$ и возвращает результат серверу. На рис. 2 представлена структурная схема взаимодействия сервера и клиентов.



MA, MB, MC

Рис.2. Реализация модели клиент/сервер

Для взаимодействия клиент-сервер использовались сокеты и RPC- механизм. Для

реализации RPC- механизма в каждом клиенте размещался пакет Client_Box с удаленным интерфейсом и описанной в нем удаленной процедурой Matrix_Multipl:

```
package Client_Box is
  pragma Remote_Procedure_Interface;
  procedure Matrix_Multipl(
    MB: in MatrixH,
    MC: in Matrix,
    MA: out MatrixH);
end Client_Box;
```

Удаленная процедура находится в клиенте, вызывается из сервера, получает данные от сервера, выполняет счет $MA_n = MB_n * MC$ и возвращает результат назад серверу.

Клиентская часть распределенного приложения оптимизирована с учетом того, что она выполняется в двухпроцессорной СМП системе. В удаленной процедуре Matrix_Multipl() создаются два процесса, которые реализуют параллельные вычисления MA_n . При этом решается задача взаимного исключения по доступу к общему ресурсу (MC) и возникающие задачи синхронизации по началу счета и возврата результата назад серверу.

Результаты исследований. Результаты исследований разделим на две группы: результат анализа эффективности использованных средств языка для программирования параллельных и распределенных вычислений и результаты экспериментов по сравнению времени выполнения разработанных приложений в зависимости от используемых программных средств и структуры PBC.

Программирование процессов. Результаты исследований показали, что язык программирования Ада95 поддерживает средства программирования параллельных процессов, позволяющие эффективно решать все проблемы, связанные с программированием процессов и организацией их взаимодействия.

Наиболее простым и эффективным средством решения задачи взаимного исключения показали себя atomic операции. Однако они не могут быть использованы в задачах взаимного исключения с условным доступом к общим ресурсом.

Реализация семафоров в языке ориентирована на синхронизацию только двух задач, поэтому не подходит для задачи взаимного исключения, которая предполагает синхронизацию нескольких задач по доступу к общему ресурсу.

Защищенные модули лишены подобных недостатков, но более сложны для программирования, требуют внимания и в целом работают медленнее остальных средств синхронизации.

В ряде случаев хорошие результаты показывало сочетание разных средств, например, прагмы Atomic и семафоров. В любом случае, входы защищенного модуля являются лучшим инструментом для задачи взаимного исключения, где доступ к общему ресурсу зависит от некоторых дополнительных условий, а защищенные функции - единственным возможным средством сокращения времени чтения общих

переменных за счет реализации в функциях параллельного (одновременного) доступа к ним.

Возможность выбора тех или иных средств синхронизации позволяет оптимальным образом подойти к их выбору для каждой конкретной задачи взаимодействия процессов.

Распределенное программирование. Удаленные подпрограммы - единственное средство, встроенные непосредственно в язык, так как сокет приходится брать из других библиотек, например Win32. Главное достоинство модели распределенных вычислений, поддерживаемое стандартом Ада95, заключается в том, что предельно упрощен переход от обычного приложения к распределенному, требующий лишь применения соответствующих прагм категорий и формирования файла конфигурации. Стандарт языка не определяет средства конфигурирования. Используемый в работе язык конфигурирования системы GLADE основан на пакете System.RPC, близок по синтаксису к языку Ада и не вызвал затруднений при формировании разделов распределенной программы и их распределения по узлам. По сравнению с другими средствами распределенного программирования (CORBA, Java/RMI), которые требуют от программиста разработки специальных программных модулей (в Java/RMI - четыре модуля), Ада модель простая, наглядная и удобная. При этом она позволяет динамическое изменение связи клиентской и серверной частей приложения.

Эксперименты. Проведенные в реальной PBC экспериментальные исследования (Таблицы 1 и 2) показали, что:

1. В целом PBC рассмотренной структуры с 1 - 8 двухпроцессорными узлами позволяет сократить время решения задачи умножения матриц в от 1,7 до 8,9 раз.
2. Наибольшее значение коэффициента ускорения 8,92 получено в PBC с 8-ю двухпроцессорными узлами при использовании механизма удаленных процедур.
3. Ада и Java распределенные приложения показали примерно одинаковые результаты с точки зрения сравнения времени выполнения распределенных и нераспределенных программ. Однако Ада приложения в целом работают медленнее, чем Java. Это можно объяснить тем, что в анализируемых Ада программах не использовалась прагма Suppress, отменяющая контроль Ада программы и сокращающая время ее выполнения. Пожертвовав надежностью, можно сократить размер выполняемого файла и уменьшить время выполнения Ада приложения.
4. Начиная с четырехузловой PBC рост значения коэффициента ускорения начинает замедляться. Причин здесь две. Первая - начинает сказываться влияние использования не самого быстрого сегодня сетевого оборудования (100 Mb Ethernet), которое не справляется с организацией взаимодействия узлов PBC. Вторая - выбранный алгоритм параллельного умножения матриц связан с рассылкой целой матрицы MC, что с увеличением числа клиентов P приводит к росту объема пересылаемой между узлами информации. Этого влияния можно избежать, если выбрать другой параллельный алгоритм умножения матриц,

позволяющий с ростом узлов РВС уменьшить объемы пересылки. При этом возрастет нагрузка на серверный узел, который теперь кроме сборки и рассылки будет вынужден заниматься и формированием результирующей матрицы. Кроме того, GLADE обеспечивает поддержку сжатия информации, пересылаемой между узлами РВС, что может служить основой для сокращения времени взаимодействия узлов.

5. При увеличении размерности матриц значения коэффициентов ускорения улучшаются. Это объясняется тем, что рост объема рассылаемой информации пропорционален N^2 , а рост вычислительных действий в каждом узле пропорционален N^3 , то есть время счета увеличивается по сравнению со временем пересылки, которое уже не так влияет на суммарное время решения рассматриваемой задачи.

Выводы. Проведенные экспериментальные исследования показали, что язык Ада95 может служить эффективным инструментом для разработки приложений для распределенных вычислительных систем с различной структурной организацией, в частности для масштабируемых РВС, в узлах которых используются СМП системы.

На кафедре вычислительной техники НТУУ- КПИ в настоящее время проводятся исследования, связанные с проектированием высокопроизводительных РВС и организации вычислительных процессов в подобных системах с использованием языков программирования, библиотек, систем планирования нагрузки узлов и др. Ближайшие работы связаны с исследованием вопросов построения мощных (кластерных) РВС с изменяемой (перестраиваемой) структурной организацией и с решением в таких системах широкого спектра сложных вычислительных задач.

ЛИТЕРАТУРА

1. Корочкин А.В. Ада95: Введение в программирование // Век, Киев, 1999. – 264 с.
2. Korochkin. D., Korochkin S. Experimental Performance Analysis of the Ada95 and Java Parallel Program on SMP Systems // Proceeding of ACM SIGAda Annual International Conference (SIGAda 2002), USA, 2002.- P.53-56.

3. Taft S., Duff R., Brukardt R., Ploedereder E. Consolidated Ada Reference Manual. Language and Standard Libraries // Springer, 2001. – P. 684.

4. Корочкин А., Ровто В., Жужель М., Синчук В. Анализ эффективности механизма сокетов в распределенных вычислительных системах // Вісник Національного Технічного Університету України - КПІ, Інформатика, управління та обчислювальна техніка, № 39. – Киев, 2002. – С. 99-104.

5. Корочкин А., Жужель М., Авдеев А. Корочкин Д. Защищенные модули как универсальный механизм синхронизации процессов в SMP – системах // Вісник Національного Технічного Університету України - КПІ, Інформатика, управління та обчислювальна техніка, № 36. – Киев, 2001. – С. 109-115.

6. Гладкий А., Корочкин Д. Использование языка Ада95 для программирования в распределенных вычислительных системах // Вісник Національного Технічного Університету України - КПІ, Інформатика, управління та обчислювальна техніка, № 33. – Киев, 2000. – С. 104-111.

7. Корочкин А., Мустафа А. Параллельные вычисления: Ада и Java // Вісник Національного Технічного Університету України - КПІ, Інформатика, управління та обчислювальна техніка, № 32. – Киев, 1999. – С. 137-146.

8. Корочкин А. Тулинов А. Экспериментальный анализ эффективности PVM и MPI в распределенных системах // Вісник Національного Технічного Університету України - КПІ, Інформатика, управління та обчислювальна техніка, № 38. – Киев, 2002. – С. 51-55.

9. Корочкин А., Осиевский А. Исследование вычислительных методами Монте-Карло в распределенных вычислительных системах // Вісник Національного Технічного Університету України - КПІ, Інформатика, управління та обчислювальна техніка, № 38. – Киев, 2002. – С. 92-99.

10. Korochkin A., Rovto V., Zhuzel M, Sinchuk V. Experimental Performance Analysis of the Ada95 Program on Distributed Systems // Proceeding of 1-st International Conference ACSN-2003, Lviv, 2002, P. 102-104.

11. Hoare C.A. Communicating Sequential Processes // Printice Hall, Engelwood Cliffs, NJ, 1985. – P. 346.

12. Andres G. Foundation of Multithreaded, Parallel and Distributed Programming // Addison-Wesley Longman, 2000. – P. 424.

Таблица 1.

Результаты экспериментальных исследований в РВС для N = 1000

Кол – во узлов РВС	Кол - во узлов-серверов	Коэффициент ускорения			
		Ада95		Java	
		Сокеты	RPC	Сокеты	RMI
2	1	1,79	1,72	1,86	1,76
3	2	3,32	3,23	3,48	3,45
4	3	5,23	5,34	5,01	5,15
5	4	5,87	6,02	5,22	5,82
6	5	6,74	7,14	6,07	7,09
7	6	7,43	7,88	7,02	7,56
8	7	7,94	8,16	7,88	8,03

Таблица 2.

Результаты экспериментальных исследований в РВС для N = 1500

Кол - во узлов РВС	Кол - во узлов - серверов	Коэффициент ускорения			
		Ада95		Java	
		Сокеты	RPC	Сокеты	RMI
2	1	1,84	1,84	1,88	1,86
3	2	3,42	3,31	3,55	3,85
4	3	5,36	5,45	5,16	5,77
5	4	6,14	6,24	5,34	6,74
6	5	7,12	7,55	6,53	8,14
7	6	8,02	8,85	7,68	8,74
8	7	8,24	8,92	8,11	8,83