УДК 681.3.066

АНАЛИЗ И ОЦЕНКА ПРОИЗВОДИТЕЛЬНОСТИ ТЕХНОЛОГИЙ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ

ГОРБАЧЕВ В.А., ГРИЦЕНКО Т.В.

Описываются результаты сравнительной оценки эффективности четырех наиболее популярных параллельных методов параллельного программирования и соответствующих библиотек: MPI, HPF, OpenMP и DVM. Приводится краткий обзор указанных выше методов, а также их моделей, используемых тестов и результатов эксперимента.

1. Введение

Целью данной работы является оценка и сравнение эффективности четырех наиболее популярных параллельных библиотек: MPI (Message Passing Interface) – интерфейс передачи сообщений, HPF (High Performance Fortran) – высокопроизводительный Фортран, OpenMP и DVM. Оценка основана на тестах NAS Parallel benchmark suite (NPB) [4], которые включают моделируемые приложения BT, SP, LU и эталонные тесты FT, CG и MG. Даются соответствующие рекомендации для различных подходов, в зависимости от количества процессоров.

Актуальность работы обусловлена необходимостью оценки производительности существующих технологий параллельного программирования, а также созданием рекомендаций по их использованию.

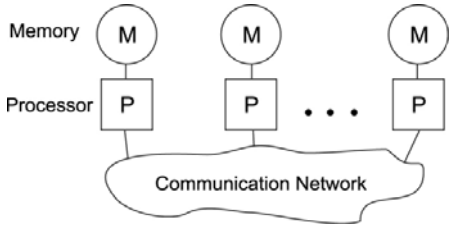
Новизна полученных результатов заключается в сравнении производительности четырех современных платформ для разработки параллельных программ в рамках задач одинаковой сложности, а также в формулировке рекомендаций по использованию этих платформ в зависимости от числа доступных процессоров или узлов вычислительной системы.

Приведем краткий обзор четырех изучаемых параллельных методов – MPI, HPF, OpenMP и DVM и их моделей.

1.1. Программная модель MPI – интерфейса передачи сообщений

Программные модели обычно классифицируют по способу использования памяти. В модели с общей памятью каждый процесс имеет доступ к общему адресному пространству, в то время как в модели передачи сообщений приложение запускается как набор автономных процессов, каждый со своей локальной памятью. В модели передачи сообщений процессы взаимодействуют с другими процессами путем отправки и приема сообщений (рисунок).

Передача сообщений широко распространена в параллельных компьютерах с распределенной памятью и в кластерах. Преимуществами модели передачи сообщений являются [9, 10]: переносимость, универсальность, т.е. модель содержит минимум ограничений относительно параллельной аппаратной платформы, простота, в том смысле, что модель поддерживает явное управление ссылками на память для упрощения отладки.



Программная модель интерфейса передачи сообщений

Первичными целями MPI являются эффективность взаимодействия процессов и переносимость. Хотя для разных систем существуют различные реализации библиотек передачи сообщений, MPI является популярной благодаря: поддержке полностью асинхронного взаимодействия, т.е. этап взаимодействия процессов может накладываться на этап вычислений, группировке процессов, т.е. процессы могут быть сгруппированы в зависимости от их контекста; переменным синхронизации, обеспечивающим безопасность и целостность передачи сообщений между процессами. При передаче и приеме сообщений синхронизация обеспечивается информацией об источнике и приемнике, маркировкой сообщений и контекстной информацией, а также о переносимости, в том смысле, что все реализации MPI основаны на стандарте, определяющем семантику для использования.

Программа MPI состоит из набора процессов и логической среды связи, объединяющей эти процессы. MPI процесс не может напрямую обратиться к памяти другого MPI процесса. Межпроцессное взаимодействие требует вызова подпрограмм в обоих процессах. В MPI определена библиотека подпрограмм, с помощью которых взаимодействуют MPI процессы. Подпрограммы библиотеки MPI обеспечивают набор функций, поддерживающих связь «точка-точка», коллективные операции, группы процессов и контексты связи, топологии процессов и операции манипуляций над типами данных. MPI содержит более 300 различных процедур для обеспечения необходимой функциональности и имеет интерфейс для языков C/C++ и Фортран.

1.2. HPF

В модели параллельной обработки данных HPF вычисления выполняются параллельно путем распределения данных по процессам. Каждый процесс обрабатывает только собственный сегмент данных. Во многих случаях компилятор HPF может обнаруживать параллельные вычисления с распределенными данными. HPF предлагает двухуровневую стратегию распределения данных. Вначале массивы должны быть сопоставлены с помощью директивы ALIGN. Затем

каждая группа сопоставленных массивов должна быть распределена между виртуальными процессорами директивой DISTRIBUTE.

Существует несколько путей указания параллелизма в HPF: с помощью определения массивов в стиле F90, операторов FORALL и WHERE, директивы INDEPENDENT и специальных средств HPF библиотеки [5]. При использовании массивов операции выполняются параллельно над сегментами данных, которыми владеет процесс.

Компилятор в случае необходимости заботится о связи данных между процессорами. Директива INDEPENDENT указывает, что между различными итерациями цикла нет никаких зависимостей и итерации могут выполняться параллельно. В частности, она указывает, что соблюдены условия Бернстейна (Bernstein's conditions): наборы читаемой и записываемой областей памяти на различных итерациях цикла не накладываются и ни одна область памяти не записывается повторно на различных итерациях цикла [6]. Все переменные цикла, которые не удовлетворяют условиям, должны быть объявлены как NEW и будут скопированы компилятором, для того чтобы цикл мог выполняться параллельно.

Компилятор предоставляет специальные средства, относящиеся к обработке распределенных массивов. Есть несколько типов таких средств: создание запросов взаимодействия, внедрение независимых циклов, создание временных и общедоступных распределенных массивов.

Внедрение запросов взаимодействия связано с обращениями к элементам, расположенным в локальной памяти разных процессов, когда они необходимы для оценки выражений с распределенными массивами или для выполнения итерации независимого цикла. Некоторые связи могут определяться на этапе компиляции, тогда как другие могут быть определены только после запуска программы, вызывая дополнительное копирование и планирование взаимодействий. В некоторых случаях вычисления могут масштабироваться, приводя к существенному замедлению. Стандарт HPF был разработан в 1993 году как расширение языка Фортран-90. Позже такие расширения были разработаны для языка C/C++.

1.3. OpenMP

OpenMP [7] разработан для поддержки переносимости параллельных программ между различными многопроцессорными архитектурами с общей памятью. OpenMP представляет собой набор директив компилятора и библиотеки вызываемых подпрограмм, расширяющий Фортран, C и C++, для поддержки параллелизма в архитектурах с общей памятью. В OpenMP используется модель параллельного выполнения fork/join. Программа начинает выполняться как один процесс, который в OpenMP называется главным потоком (master thread). Этот процесс выполняется последовательно до тех пор, пока он не дойдет до первой параллельной конструкции (области программы, заключенной между парой директив PARALLEL и END PARALLEL). Затем главный поток создает «команду» (team) потоков, включая в нее и себя.

Часть программы, заключенная в параллельную конструкцию, исполняется параллельно каждым потоком «команды» до тех пор, пока не встретится конструкция разделения работ. Директивы PARALLEL DO или DO являются таким конструкциями разделения работ, распределяющими рабочую нагрузку цикла DO между членами текущей «команды». Необходимая синхронизация выполняется в конце цикла DO, если не предусмотрена директива END DO NOWAIT. Факт общего доступа к данным, переменным или массивам устанавливается в начале параллельных конструкций или конструкций разделения работ путем использования ключей SHARED и PRIVATE. После завершения параллельной конструкции потоки в «команде» синхронизируются, и выполнение продолжает только главный поток.

OpenMP предлагает мощную концепцию Orphan-директив, которые позволяют при минимальных изменениях обычных последовательных программ организовать параллельное выполнение самых трудоемких частей программы.

Orphan-директивы расположены вне лексического пространства соответствующей параллельной конструкции. Данная концепция позволяет пользователю контролировать управление или синхронизацию из любой части параллельной конструкции.

1.4. DVM

DVM является расширением языков ANSI-C и Фортран с помощью аннотаций, называемых DVM-директивами. DVM-директивы могут быть условно разделены на три подмножества: директивы распределения данных, директивы распределения вычислений и спецификации удаленных данных. Модель параллелизма в DVM основана на определенной форме параллелизма данных, называемой SPMD (Single Program, Multiple Data). В этой модели все процессоры исполняют одну и ту же программу, но каждый процессор выполняет свое собственное подмножество инструкций в соответствии с распределением данных. В модели DVM пользователь вначале определяет многомерное распределение виртуальных процессоров, на секции которых будут отображены данные и вычисления. Секция может состоять их целого набора процессоров или только из одного процессора. Затем определяются массивы (distributed data), распределяемые между процессорами. Эти массивы определяются директивами отображения данных. Остальные переменные, распределяемые по умолчанию, отображаются по одной копии на каждый процессор (replicated data). Значение копируемых данных должно быть одинаковым на всех задействованных процессорах. Единственным исключением из этого являются переменные в параллельных циклах. Отображение данных определяет набор локальных (собственных) перемен-

ных для каждого процессора. Набор собственных переменных определяет правило собственных вычислений: процессор присваивает значения только собственным переменным. DVM модель определяет два уровня параллелизма: параллелизм данных и параллелизм задач. Первый осуществляется распределением тесно связанных итераций циклов между процессорами. Итерация цикла выполняется полностью на одном процессоре. Инструкции, расположенные вне параллельного цикла, выполняются согласно правилу собственных вычислений. Параллелизм задач осуществляется распределением данных и вычислений между разъединенными секциями процессоров. При вычислении значений собственных переменных процессору могут понадобиться значения как собственной, так и удаленной переменных. Все удаленные переменные должны быть определены при помощи директив удаленного доступа.

2. NAS Parallel Benchmarks

Тесты NAS Parallel Benchmarks (NPB) [4] были получены на основе кодов для Computational Fluid Dynamics (CFD) программ. Они разработаны для сравнения работы параллельных компьютеров и получили широкое признание как стандартный показатель производительности компьютера. NPB состоит из пяти ядер и трех моделируемых CFD приложений, полученных из нескольких классов аэрофизических приложений. Эти тесты являются вычислительным ядром пяти численных методов, используемых в CFD приложениях. Моделируемые CFD приложения воспроизводят большую часть преобразований над данными и вычислений из исходных CFD кодов. В этой статье эталонные тесты определены только алгоритмически, подробности о пакете тестов NPB можно найти в [4]. В данной статье рассмотрены только шесть тестов (исключая IS и EP):

· **BT** является моделируемым CFD приложением, использующим неявный алгоритм для решения трехмерного (3-D) уравнения

$$Ku = r, \qquad (1)$$

где u и r – векторы, размерностью 5x1, определенные в вершинах трехмерной прямоугольной решетки; K – блочно-диагональная матрица, состоящая из 7 блоков 5x5. Конечноразностное решение основано на Alternating Direction Implicit (ADI) аппроксимирующей факторизации, разделяющей размерности x, y и z: $K \cong BT_x \cdot BT_y \cdot BT_z$, где BT_x, BT_y и BT_z являются блочно-диагональными матрицами из трех блоков 5x5. Результирующая система затем решается путем вычмсления блочных треугольных систем в каждом из направлений *x*, *y* и z.

· **SP** является моделируемым CFD приложением со структурой, подобной BT. Конечноразностное решение проблемы основано на Beam-Warming аппроксимирующей факторизации и Pulliam-Chaussee диагонализации оператора в уравнении (1) и добавляет искусственное разложение четвертого порядка:

$$K \cong T_x \cdot P_x \cdot T_x^{-1} \cdot T_y \cdot P_y \cdot T_y^{-1} \cdot T_z \cdot P_z \cdot T_z^{-1},$$

где Tx, Ty и Tz – блочные диагональные матрицы из блоков 5x5; Px, Py и Pz – скалярные пентадиагональные матрицы.

Результирующая система решается путем инвертирования блочных диагональных матриц $T_x, T_x^{-1} \cdot T_y, T_y^{-1} \cdot T_z$ и последующего решения скалярных пентадиагональных систем.

· **LU** является моделируемым CFD приложением, использующим Symmetric Successive Over-Relaxation (SSOR) метод для решения блочно-диагональной системы, полученной в результате конечноразностной дискретизации уравнений Navier-Stokes в трехмерном пространстве разбиением на блоки верхней и нижней треугольных систем:

$$K \cong \omega(2-\omega)(D+\omega Y)(I+\omega D^{-1}Z),$$

где ω – параметр релаксации; D – главная блочная диагональ; K, Y состоит из трех подблочных диагоналей и Z состоит из трех суперблочных диагоналей. Задача решается путем вычисления элементов треугольной матрицы и решением верхней и нижней треугольных системы.

· **FT** состоит из вычислительного ядра трехмерного спектрального метода, основанного на быстром преобразовании Фурье. FT выполняет три одномерных (1-D) быстрых преобразования Фурье (одно для каждого измерения). Преобразование формулируется как матрично-векторное умножение:

$$v = (F_m \otimes F_n \otimes F_k)u,$$

где u и v – трехмерные массивы размерностью (m,n,k), представленные как векторы размерностью mxnxk. $A \otimes B$ является блочной матрицей с блоками $a_{ij}B$ и называется тензорным значением A и B. Алгоритм основан на представлении матрицы быстрого преобразования Фурье как произведения трех матриц, выполняющих несколько быстрых преобразований Фурье в одном направлении. Далее FT выполняет быстрые преобразования Фурье в x-, y- и z- направлениях. Ядро быстрого преобразования Фурье реализовано как Swarztrauber векторизация алгоритма сортировки, выполняющего независимые быстрые преобразования Фурье над множеством векторов.

· **MG** выполняет итерации V-цикличного мультирешеточного алгоритма для решения уравнения Пуассона $\nabla u = v$ для трехмерной решетки с периодическими граничными условиями [4]. Каждая итерация состоит из оценки остатка $r = v - Au$ и коррекции: $u = u + Mr$, где M – V-цикличный мультирешеточный оператор.

· **CG** использует метод сопряженного градиента (**C**onjugate **G**radient Method) для вычисления аппроксимации наименьшего экстремума большой, редкой, неструктурированной матрицы (sparse matrix). Единичная итерация может быть записана следующим образом:

$$q = Ap,\ d = p^T q,\ \alpha = \frac{\rho}{d},$$

$$z = z + \alpha p,\ r = r - \alpha q,\ \rho_0 = \rho,$$

$$\rho = r^T r,\ \beta = \frac{\rho}{\rho_0},\ p = r + \beta p \quad .$$

Наиболее трудоемкой операцией является матрично-векторное умножение $q = Ap$, выполняемое параллельно.

3. Результаты тестов

Реализация NPB основана на стандарте MPI. Таким образом, возможно сравнение MPI-версии с реализациями NPB средствами HPF, OpenMP и DVM технологий. MPI, OpenMP и HPF версии тестировались на аппаратной платформе Origin 2000, а DVM на RCC-кластере [3]. Результаты, приведенные в табл.1-3, основаны на информации, полученной из источников [1-3,8]. В таблицах приведено время выполнения MPI, OpenMP, HPF и DVM версий шести тестов, а также ускорение времени выполнения (2) различных версий для каждого теста в зависимости от количества задействованных процессоров:

$$\text{Speedup} = T_n / T_S, \qquad (2)$$

где T_n – время выполнения на мультипроцессорном компьютере (n = 2, 4, 8, 16, 32) и T_S – время выполнения на одном процессоре.

Таблица 1

Время выполнения MPI-, OpenMP- и HPF-версий NPB тестов на платформе Origin 2000

Тест	Техн	1	2	4	8	16	25	32	36
BT	MPI	1641.2	770	411.6	-	93.7	62.1	50	48.5
	OpenMP	1477.8	749	387.6	-	106.1	76	54.3	53.7
	HPF	2975	1479.4	714.4	-	179.7	149.6	108.4	
SP	MPI	1638.4	700	412.8	178	88.4	55.2	45	36.8
	OpenMP	1227.1	646	350.4	175	91.4	72.7	51.8	-
	HPF	2894.9	1489	626.5	267.7	138.2	115.5	97	-
LU	MPI	1548.4	731.1	344	158.7	73.9	-	38.9	-
	OpenMP	1234.4	585.6	336.2	184.4	96.1	-	57.1	-
	HPF	3512.2	2142.3	1046.3	567.4	311.8	-	230.6	-
FT	MPI	133.4	82.5	41.3	21.4	11.3	-	6.1	-
	OpenMP	114.6	60.22	30.94	16.15	8.59	-	4.84	-
	HPF	174.4	95	44.3	23.2	12.9	-	8.2	-
MG	MPI	53.4	31	14.9	7.5	4	-	2.2	-
	OpenMP	47.58	27.06	13.71	6.99	3.91	-	2.53	-
	HPF	143.6	119.9	82.8	46.9	25.3	-	16.4	-
CG	MPI	44.4	28.46	13.62	4.54	2.67	-	1.78	-
	OpenMP	47.74	28.65	14.03	4.76	2.46	-	2.19	-
	HPF	54	30.6	14.9	6.8	5.9	-	10	-

Таблица 2

Время выполнения MPI- и DVM-версий NPB тестов на RCC-кластере

Тест	Техн	1	2	4	8	16
BT	MPI	2450	1300	606.1	284.7	220.8
	DVM	2548.5	1584.3	712.3	380.6	231.2
SP	MPI	1670.7	-	435.2	207.7	146.5
	DVM	2132.2	-	616.6	311.7	201.6
LU	MPI	1581.5	989.5	361.5	165.7	84.5
	DVM	1886	974.4	512.3	265.9	143.2
FT	MPI	130.2	88.2	42.5	21.2	13.3
	DVM	136.1	75.8	42.6	26	14.5
MG	MPI	77.7	47.9	22.2	9.7	7
	DVM	71.5	36.5	18.8	10.5	6.7
CG	MPI	41.4	28.3	11.7	6.3	5
	DVM	42.9	22.8	13.6	9.1	7

Таблица 3

Ускорение времени выполнения MPI-, OpenMP-, HPF- и DVM-версий NPB тестов

Тест	Техн	1	2	4	8	16	25	32	36
BT	MPI	1	1.9	3.98	-	17.52	26.43	31	33.84
	OpenMP	1	1.97	3.81	-	13.93	19.44	27.22	27.52
	HPF	1	2.01	4.16	-	16.55	19.89	27.44	-
	DVM	1	1.75	3.58	-	11.02	-	-	-
SP	MPI	1	2	3.97	8.5	10.61	18.53	29.68	29.85
	OpenMP	1	1.9	3.5	7.01	8.31	13.42	16.88	23.69
	HPF	1	1.94	4.62	10.81	11.13	20.95	25.06	29.84
	DVM	1	1.7	3.46	6.65	6.84	10.58	-	-
LU	MPI	1	2.12	4.5	9.76	20.95	-	39.8	-
	OpenMP	1	2.11	3.67	6.7	12.84	-	21.61	-
	HPF	1	1.64	3.36	6.19	11.26	-	15.23	-
	DVM	1	1.94	3.68	7.09	13.17	-	-	-
FT	MPI	1	1.62	3.23	6.23	11.8	-	21.86	-
	OpenMP	1	1.9	3.7	7.08	13.34	-	23.65	-
	HPF	1	1.83	3.94	7.51	13.52	-	21.23	-
	DVM	1	1.79	3.19	5.23	9.38	-	-	-
MG	MPI	1	1.72	3.58	7.12	13.35	-	24.27	-
	OpenMP	1	1.76	3.47	6.81	12.17	-	18.81	-
	HPF	1	1.19	1.73	3.06	5.67	-	8.75	-
	DVM	1	1.96	3.8	6.81	10.67	-	-	-
CG	MPI	1	1.56	3.26	9.78	16.63	-	24.94	-
	OpenMP	1	1.66	3.4	10.03	19.41	-	21.79	-
	HPF	1	1.76	3.62	7.94	9.15	-	5.4	-
	DVM	1	1.88	3.15	4.71	6.13	-	-	-

4. Заключение

В большинстве случаев время выполнения MPI-версии является меньшим по сравнению с остальными подходами. OpenMP-версия приблизительно на 10% медленнее, чем MPI-версия, в то время как DVM медленнее на 20% и HPF – на 30%. Разрыв во времени выполнения увеличивается с количеством процессоров. Это происходит благодаря более эффективной модели распределения памяти в MPI по сравнению с остальными технологиями. Коэффициент ускорения

времени выполнения MPI-версии также выше, чем при других подходах. Таким образом, если задача распределена более, чем между 16 процессами, MPI производит самое быстрое решение для используемого набора тестов. Однако MPI – низкоуровневый язык в терминах параллельного программирования. Это "ассемблер" для параллельного программирования, и наиболее трудоемкой частью является этап распределения данных и построения схемы связей между процессами. В некоторых случаях, когда число процессоров меньше 16, рекомендуется применять другие технологии, такие как OpenMP или DVM, которые, хотя и уступают по скорости MPI, являются более легкими для использования и требуют гораздо меньше временных затрат на разработку.

Сравнение на базе NPB 2.3 тестов не может быть всесторонним, так как эти тесты разработаны на очень высоком уровне командой экспертов, но оно может дать общее представление о возможностях рассмотренных параллельных технологий.

Практическая значимость. Параллельные программы требуют значительных модификаций на этапе разработки и использования для оптимизации вычислительных показателей. В этой связи результаты, полученные в работе, могут быть использованы как практические рекомендации разработчику параллельных приложений.

Литература: 1.*Michael Frumkin*, Haoqiang Jin and Jerry Yan (1998). Implementation of NAS Parallel Benchmarks in High Performance Fortran. NAS Technical Report NAS-98-009, NASA Ames Research Center, 10 p. **2.** *Jin H., Frumkin M. and J. Yan* (1998). The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance. NASA Ames Research Center, 26 p. **3.** *Крюков В.А.* Разработка параллельных программ для вычислительных кластеров и сетей. 2002. Институт прикладной математики им. М.В. Келдыша, РАН. 22 с. **4.** *Bailey D., Harris T., Sahpir W., van der Wijngaart R., Woo A., Yarrow M.* December 1995. The NAS Parallel Benchmarks 2.0. Report NAS-95-020. 24 p. **5.** *Koelbel C.H.* November 1997. An Introduction to HPF 2.0. High Performance Fortran – Practice and Experience. Supercomputing 97. 27 p. **6.** *Koelbel C.H., Loverman D.B., Shreiber R., Steele Jr. GL., Zosel M.E.* 1994. The High Performance Fortran Handbook. MIT Press. 262 p. **7.** *OpenMP* Fortran Application Program Interface, http://www.openmp.org. **8.** *DVM*. Execution performance of NAS tests, http://www.keldysh.ru/dvm/. **9.** *Writing* Message-Passing Parallel Programs with MPI. http://www.epcc.ed.ac.uk/computing/training/document_archive/mpi-course/mpi-course.book_1.html. 10.HP MPI User's Guide. Fourth Edition http://www.docu.sd.id.ethz.ch/comp/stardust/SW/mpi/title.html.

Поступила в редколлегию 05.12.2006

Рецензент: д-р техн. наук, проф. Кривуля Г.Ф.

Горбачев Валерий Александрович, профессор каф. ЭВМ ХНУРЭ. Научные интересы: моделирование систем. Хобби: компьютеры, волейбол, автомобили. Адрес: Украина, 61166, Харьков, пр.Ленина, 14, тел. +38(057)7021-115.

Гриценко Тарас Васильевич, ааспирант каф. ЭВМ ХНУРЭ. Научные интересы: моделирование систем, программирование,параллельные вычисления. Хобби: компьютеры, автомобили. Адрес: Украина, 61166, Харьков, пр.Ленина, 14, тел. +38(057)7021-354.

УДК519.713

ПРОБЛЕМЫ АНТИВИРУСНОЙ ИНДУСТРИИ, МЕТОДЫ БОРЬБЫ С КОМПЬЮТЕРНЫМИ УГРОЗАМИ И БЛИЖАЙШИЕ ПЕРСПЕКТИВЫ РАЗВИТИЯ

ГОРОБЕЦ А.А., КУНИЦКИЙ А.В., ПАРФЕНТИЙ А.Н., ЧУВИЛО О.А.

Исследуется состояние мирового рынка антивирусной индустрии, применяемые технологии борьбы с различными компьютерными угрозами, выявляются проблемы, стоящие перед создателями антивирусных продуктов, и тенденции развития средств борьбы с компьютерными угрозами на ближайшую перспективу.

1. Введение

Цель работы – исследовать рынок средств борьбы с вирусной угрозой, сами антивирусные технологии для выявления в них узких мест (недостатков, проблем), что является необходимым для совершенствования этих технологий.

Задачи исследования – выявить приоритетные направления развития антивирусных технологий и ближайшие перспективы развития антивирусной индустрии.

Объективно сложившаяся обстановка в сфере функционирования компьютерных технологий такова, что постоянные вирусные и троянские атаки терроризируют практически всех пользователей Internet – домашних, небольшие и средние компании, глобальные корпорации и государственные структуры и таким образом оказывают серьезное деструктивное влияние на всю мировую экономику. В настоящий момент уже ясно, что основной целью современных создателей вирусов является извлечение нелегальной прибыли путем создания и распространения вредоносных программ, с помощью которых происходит: воровство частной и корпоративной банковской информации (получение доступа к банковским счетам персональных пользователей и организаций); воровство номеров кредитных карт; распределенные сетевые атаки (DDoS-атаки) с последующим требованием денежного выкупа за прекращение атаки (современный компьютерный вариант обычного рэкета); создание сетей троянских прокси-серверов для рассылки спама (и коммерческое использование этих сетей); создание зомби-сетей для многофункционального использования; создание программ, скачивающих и устанавливающих системы показа нежелательной рекламы; внедрение в компьютеры троянских программ, постоянно звонящих на платные (и весьма дорогие) телефонные номера; иные действия, связанные с возможным