

УДК 004.43

О.Н. Паулин, канд. техн. наук,  
Т.И. Усова, аспирант

## **СРАВНИТЕЛЬНАЯ ХАРАКТЕРИСТИКА ТЕХНОЛОГИЙ И ЯЗЫКОВ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ ДЛЯ ВЫЧИСЛИТЕЛЬНОГО КЛАСТЕРА**

*Порівнюються характеристики найбільш розповсюджених технологій та мов паралельного програмування (C/C++, Java, Ada, Lisp(NewLisp), API OpenMP, API MPI) з метою створення інформаційної технології розпаралелювання розв'язку нелінійних рівнянь на обчислювальному кластері*

*Сравниваются характеристики наиболее распространенных технологий и языков параллельного программирования (C/C++, Java, Ada, Lisp(NewLisp), API OpenMP, API MPI) с целью создания информационной технологии распараллеливания решения нелинейных уравнений на вычислительном кластере.*

*Are compared characteristic of the most widespread technology and parallel language (c/c++, java, ada, lisp(newlisp), api openmp, api mpi) for the purpose of creation information technology of parallel solving of nonlinear equations on the compute cluster.*

### **Введение**

В последнее время появилось все больше задач, требующих значительных вычислительных затрат на свою реализацию. Это могут быть задачи как обработки большого объема данных, так и сложных длительных вычислений. В обоих случаях для их решения использование обычного ПК довольно затруднительно. В связи с этим было найдено несколько методов их оптимизации. Например, существует возможность использования многопроцессорных ЭВМ или их близкого «родственника» – кластеров. Для использования обеих систем применяют несколько различных программных средств. Параллельному программированию в настоящее время уделяется немалое внимание. Ведутся разработки по следующим направлениям: функциональное параллельное программирование, системы граф-схемного потокового программирования, системное программное обеспечение вычислительных кластерных систем и многие другие. Увеличилось разнообразие средств для разработки параллельных программ. Это связано как с развитием архитектурных решений процессоров, так и с развитием языков программирования.

Решающими вопросами при выборе языка программирования (ЯП) являются: какая архитектура доступа к памяти будет использована, насколько хорошо распараллеливается алгоритм, какой объем разделяемых данных должны иметь процессы или нити и как часто им нужно синхронизовать эти данные.

**Постановка задачи:** сравнить технологии и языки параллельного программирования и выбрать наиболее подходящее программное средство для реализации ИТ распараллеливания решения нелинейных уравнений на вычислительном кластере.

### **Архитектуры доступа к памяти**

Существуют следующие архитектуры доступа к памяти [1]:

1. **SMP** (symmetric multiprocessing) – симметричная многопроцессорная архитектура. Главной особенностью систем с архитектурой SMP является наличие общей физической памяти, разделяемой всеми процессорами. Память служит, в частности, для передачи сообщений между процессорами, при этом все вычислительные устройства при обращении к ней имеют равные права и одну и ту же адресацию для всех ячеек памяти. Поэтому SMP-архитектура называется симметричной. Это преимущество позволяет эффективно обмениваться данными с другими вычислительными устройствами.

Важным преимуществом SMP-систем является простота и универсальность для программирования. Архитектура SMP не накладывает ограничений на модель программирования, используемую при создании приложения: обычно используется модель параллельных ветвей, когда все процессоры работают независимо один от другого. Однако можно реализовать и модели, использующие межпроцессорный обмен. Использование общей памяти увеличивает скорость такого обмена, пользователь также имеет доступ сразу ко всему объему памяти.

Основным недостатком систем с общей памятью является их плохое масштабирование.

Эту архитектуру поддерживают все неспециализированные языки, такие как Delphi, C/C++, Java, Ada, Lisp (NewLisp), API OpenMP.

2. Массивно-параллельная архитектура – *MPP* (massive parallel processing), куда можно отнести и кластерные системы. Главная особенность такой архитектуры состоит в том, что память физически разделена. В этом случае система строится из отдельных модулей, содержащих процессор, локальный банк операционной памяти (ОП), коммуникационные процессоры или сетевые адAPTERы, иногда – жесткие диски и/или другие устройства ввода/вывода.

По сути, такие модули представляют собой полнофункциональные компьютеры. Доступ к банку ОП из данного модуля имеют только процессоры (ЦП) из этого же модуля. Модули соединяются специальными коммуникационными каналами. Пользователь может определить логический номер процессора, к которому он подключен, и организовать обмен сообщениями с другими процессорами.

Главным преимуществом систем с раздельной памятью является хорошая масштабируемость. В отличие от SMP-систем, в машинах с раздельной памятью каждый процессор имеет доступ только к своей локальной памяти, в связи с чем не возникает необходимости в потактовой синхронизации процессоров.

Недостаток – отсутствие общей памяти заметно снижает скорость межпроцессорного

обмена, поскольку нет общей среды для хранения данных, предназначенных для обмена между процессорами, и требуется специальная техника программирования для реализации обмена сообщениями между процессорами. Кроме того, каждый процессор может использовать только ограниченный объем локального банка памяти.

Данную архитектуру поддерживают C/C++, Java, API MPI.

3. Гибридная архитектура – *NUMA* (nonuniform memory access) – архитектура с неоднородным доступом к памяти, когда память физически распределена по различным частям системы, но логически она является общей, так что пользователь видит единое адресное пространство.

Система построена из однородных базовых модулей (плат), состоящих из небольшого числа процессоров и блока памяти. Модули объединены с помощью высокоскоростного коммутатора. По существу, архитектура NUMA является MPP архитектурой, где в качестве отдельных вычислительных элементов берутся SMP узлы. Доступ к памяти и обмен данными внутри одного SMP-узла осуществляется через локальную память узла и происходит очень быстро, а к процессорам другого SMP-узла тоже есть доступ, но более медленный и через более сложную систему адресации. Поддерживают эту архитектуру языки программирования Linda и API OpenMP.

Язык Java благодаря наличию своей виртуальной машины стоит особняком, так как с ее помощью он становится практически архитектурно-независимым.

### Графические процессоры

В последнее время получили большое развитие графические процессоры как многопроцессорные системы, которые обладают специализированной конвейерной архитектурой [2].

На современных видеокартах содержится несколько мультипроцессоров, каждый из них обладает восемью-десятью ядрами и сотнями ALU в целом, несколькими тысячами регистров и небольшим объемом разделяемой общей памяти. Кроме того, видеокарта имеет быструю глобальную память с доступом к ней всех мультипроцессоров,

локальную память в каждом мультипроцессоре, а также специальную память для констант.

У всех GPU обычно есть по несколько контроллеров. Кроме того, на видеокартах применяется более быстрая память, в результате чего видеочипам доступна большая пропускная способность памяти, что весьма важно для параллельных расчётов со значительными потоками данных.

Обычно CPU исполняет 1-2 потока вычислений на одно процессорное ядро, а видеочипы могут поддерживать до 1024 потоков на каждый мультипроцессор, которых в чипе несколько. И если переключение с одного потока на другой для CPU состоит из сотни тактов, то GPU переключает несколько потоков за один такт.

Кроме того, центральные процессоры используют SIMD (одна инструкция выполняется над многочисленными данными) для векторных вычислений, а видеочипы применяют SIMT (одна инструкция и несколько потоков) для скалярной обработки потоков. SIMT не требует, чтобы разработчик преобразовывал данные в векторы, и допускает произвольные ветвления в потоках.

Для того чтобы использовать все эти возможности в собственных целях применяется техника использования графического процессора видеокарты для общих вычислений, обычно проводимых центральным процессором – GPGPU (General-purpose graphics processing units — «GPU общего назначения»). Была создана технология CUDA –программно-аппаратная вычислительная архитектура, основанная на расширении языка Си, которая позволяет организовать доступ к набору инструкций графического ускорителя и управления его памятью. Это упрощает реализацию параллельных вычислений на GPU как на обычном вычислительном устройстве без необходимости использования графических API. Кроме того, уже существуют компиляторы CUDA для Fortran и C/C++, CUDA поддерживает следующие языки: C/C++, Matlab, Fortran, Python, .NET, F#, Java [9].

Основные направления, в которых сейчас применяются вычисления на GPU:

анализ и обработка изображений и сигналов, симуляция физики, вычислительная математика и т.д.

Таким образом, на данный момент можно практически на любом языке написать программу для любой архитектуры доступа к памяти.

### **Технология взаимодействия параллельных процессов**

*Синхронизация процессов и данных.* Как уже говорилось, для любой параллельной программы актуальны вопросы синхронизации процессов и данных, поэтому большое внимание должно быть уделено выбору языка программирования для реализации любой информационной технологии. Проведем сравнительную характеристику для некоторых из языков, используемых в параллельном программировании.

*Синхронизация процессов.* При синхронизации процессов могут использоваться средства межпроцессного взаимодействия [12]. Среди наиболее часто применяемых средств – сигналы и сообщения, семафоры и мьютексы, каналы, randеву, совместно используемая память (в C/C++, Java, Ada, Lisp(NewLisp), Occam) [4,3,6,10,13]. Следует упомянуть и высокоуровневый механизм взаимодействия и синхронизации процессов, обеспечивающий доступ к неразделяемым ресурсам – монитор (используется в Java, его подобие со своими особенностями – в Ada) [4, 10].

Существует такое средство синхронизации как барьер. При создании барьера указывается количество нитей N, необходимое для перехода через барьер. Если количество нитей, ожидающих возле барьера, меньше N-1, нить блокируется. Когда набирается N нитей, все они разблокируются и продолжают исполнение. Барьеры полезны для синхронизации нитей, выполняющих части одной и той же работы, например параллельные вычисления. Они позволяют гарантировать, что даже при неравномерной загрузке системы нити будут выполнять равный объем работы в единицу времени [8]. Барьеры реализованы в таких API, как OpenMP и MPI [3, 8].

*Синхронизация данных.* Программная синхронизация данных обычно осуществля-

ется двумя методами: либо с помощью сообщений, либо с помощью разделяемой памяти.

Программы с разделяемой памятью удобны, когда нити алгоритма должны часто обмениваться большими объемами данных и/или осуществлять произвольный доступ к разделяемым данным большого объема. Программы, обменивающиеся сообщениями, удобны, когда объем разделяемых данных невелик, а эти данные изменяются редко и в предсказуемых местах.

Наиболее распространенная технология параллельных вычислений с разделяемой памятью – OpenMP. OpenMP-программа в начале блока разветвляется на несколько нитей, а в конце блока собирает результаты исполнения этих нитей. Таким образом, параллелизм программы имеет блочную структуру, аналогичную структуре кода в C и Java. Это резко сужает количество возможных программ, но в то же время это значительно упрощает разработку и отладку.

Наиболее распространенная технология разработки параллельных вычислительных программ с обменом сообщениями так и называется – MPI (Message Passing Interface, интерфейс передачи сообщений). Она представляет собой библиотеку, реализующую высокоровневый обмен типизированными сообщениями между процессами. Передача сообщений может осуществляться как по сети, так и через локальные средства IPC. Также предоставляется имитация семафоров, барьеров и разделяемой памяти при помощи сообщений. MPI обеспечивает запуск многопроцессных вычислительных приложений как на одном компьютере (даже однопроцессорном – это может быть полезно при отладке), так и на многомашинных вычислительных кластерах. Для большинства задач необходимо, чтобы эти машины были соединены высокопроизводительными сетевыми интерфейсами.

В последние годы возник интерес к гибридным приложениям MPI/OpenMP. Действительно, в современных вычислительных кластерах часто используются многопроцессорные машины. Кроме того, приложения MPI часто проводят довольно много времени в ожидании обмена данными; в это время

машина могла бы заниматься исполнением другой части работы. На некоторых задачах использование гибридной вычислительной модели позволяет достичь улучшения на десятки процентов и даже в разы по сравнению с чистыми MPI и OpenMP.

Для задач, которые нуждаются в редком обмене сообщениями небольшого объема, интересны вычислительные сети типа GRID. Такие сети представляют собой обычные персональные компьютеры, соединенные обычной офисной локальной сетью или даже через Интернет. Эти сети часто допускают произвольное подключение и отключение машин. В этом случае GRID система позволяет использовать персональные компьютеры во время простоя.

Так, разделяемую память используют Linda, Ada, Lisp (NewLisp) [3,4,6], а сообщениями обмениваются Occam и API MPI. Языки C/C++ и Java поддерживают оба варианта обмена данными. В языке Linda нет явных средств синхронизации. Она реализуется программистом с учетом поставленной задачи.

#### *Поддержка создания потоков и процессов*

Параллельность программ обычно обеспечивается созданием внутри них дочерних процессов или потоков, которые могут работать параллельно на нескольких процессорах. Простейшей реализацией такого подхода для языков программирования является возможность создания дочерних процессов, которые используют общие ресурсы операционной системы (ОС), и управление которыми осуществляется ОС. Этот подход существует в таких языках, как C/C++, Linda, Lisp (NewLisp), Occam [3,6,13 ].

Немного сложнее реализация потоков. Все потоки процесса разделяют общие ресурсы. Изменения, вызванные одним потоком, немедленно становятся доступны другим. Наличие потоков может ускорить программу, так как происходит экономия времени на создание, завершение, переключение и обмен данными между потоками в пределах одного процесса. Таким образом, здесь уже имеют место процессы, потоки, имеющие доступ к ресурсам своего процесса, и, непосредственно, ресурсы ОС.

Но следует учесть, что поддержка потоков полностью осуществляется ОС. Поэтому были разработаны встроенные библиотеки для поддержки потоков. Они существуют практически во всех языках высокого уровня: C/C++, Java, Ada, API OpenMP[3, 4, 6, 10, 13].

В API MPI реализация немного отличается. Дело в том, что MPI-процесс может быть многонитевым, но сделать MPI-вызовы разрешено только тому процессу, который проводил инициализацию MPI.

Далее рассмотрим некоторые языки программирования более подробно с точки зрения особенностей программирования параллельных программ.

#### *Технология OpenMP*

Для многопоточных приложений, работающих в параллельных вычислительных системах с общей памятью, был создан специальный API (Application programming interface) – OpenMP. Этот API может использоваться в компиляторах Fortran и C/C++ для осуществления возможности автоматического распараллеливания.

Принцип реализации параллельных вычислений основан на том, что главный поток создает набор подчиненных потоков и задача распределяется между ними. Задачи и данные для параллельных приложений описываются специальными директивами соответствующего языка. Количество создаваемых потоков регулируется или самой программой, или с помощью переменных окружения. Для взаимодействия потоков в этом интерфейсе прописано шесть директив синхронизации: master ... end master – (определяет блок кода, выполняемый только мастером, т.е. нулевой нитью; critical ... end critical – устанавливаем критическую секцию, т.е. блок кода, который не должен выполняться одновременно двумя или более нитями; barrier – определяет точку барьерной синхронизации; atomic – переменную, корректно обновляемую несколькими нитями; ordered ... end ordered – блок внутри тела цикла, выполняемым в том порядке, в котором итерации идут в последовательном цикле; flush – явно определяет точку, в которой реализация должна обеспечить одинаковый вид памяти для всех нитей.

Параллельные приложения, созданные для многопроцессорных систем при помощи интерфейса OpenMP, должны удовлетворять следующему требованию: количество потоков не должно превышать количество процессоров. Программы OpenMP исполняются как несколько потоков в рамках одного процесса. Это практически исключает возможность исполнения таких программ – во всяком случае, в чистом виде – на много-машинных комплексах. Тем не менее, существуют вычислительные комплексы с большим количеством процессоров, предоставляющие разделяемую память, пригодную для исполнения многопоточных программ – так называемые NUMA-системы.

#### *Технология MPI*

Для написания параллельных программ может также использоваться распространенный API – MPI, интерфейс передачи сообщений. Принцип действия в данном интерфейсе основан на обмене сообщениями между процессами при помощи коммуникаторов (под коммуникатором в MPI понимается специально создаваемый служебный объект, объединяющий в своем составе группу процессов и ряд дополнительных параметров (контекст), используемых при выполнении операций передачи данных).

Существуют собственные средства синхронизации, позволяющие приостанавливать обмен данными в критические моменты. Также как и для OpenMP, реализации MPI существуют для Fortran и C/C++. Стандарты MPI поддерживаются как на платформе ОС, так и в приложениях пользователя. В вычислительных задачах MPI/OpenMP применяется примитив синхронизации – барьер. Существуют также примеры его применения в других задачах, но, как правило, это также задачи вычислительного характера. MPI используется при разработке приложений для кластеров и суперкомпьютеров. В связи с широким распространением MPI ведутся достаточно активные исследования по его применению. Например, исследуется совместное использование стандарта MPI и нитей POSIX для организации обменов сообщениями в кластерных вычислительных системах, реализация привязки MPI-процессов к многоядерным

узлам вычислительных систем, повышение производительности коллективных операций MPICH-2 и другие.

### **Языки параллельного программирования.**

#### *C/C++ и Java*

Это наиболее распространенные на данный момент языки программирования. Они являются сложными для сравнения, так как Java – С-подобный объектно-ориентированный язык.

В обоих языках реализация многозадачности основана на выделении операционной системой квантов времени для каждого процесса. Эти кванты распределяются между процессами с учетом приоритетов последних. В языке C/C++ многозадачность реализуется с помощью API-функций ОС. В Java многозадачность поддерживается на уровне языка – часть примитивов синхронизации встроена в систему реального времени и все системные библиотеки Java созданы таким образом, что могут быть использованы в многозадачных приложениях, при этом классы могут подгружаться динамически и с любого узла сети.

Так как Java разрабатывался с учетом того, что на нем могут создаваться параллельные приложения, работа с ними упрощена существованием монитора, позволяющего выбирать процесс или поток, который должен участвовать в многопоточной задаче, и далее монитор берет управление на себя. В то время как в языке C/C++ программист должен вручную прорабатывать и прописывать все действия процессов и потоков, рискуя запутаться в них. И в языке C/C++, и в языке Java существует понятие приоритетов, когда процесс с более высоким приоритетом получает большее число квантов времени для своей работы, однако и процесс с наименьшим приоритетом успеет получить хотя бы один квант времени до того, как более приоритетный процесс закончит свою работу. Если сравнить C/C++ и Java по возможности присвоения приоритетов созданным процессам, то можно увидеть, что в C/C++ есть возможность создавать процессы с любым приоритетом от наименьшего до наивысшего из существующих в системе, а в Java номер приоритета выбирается из

выделенного в виртуальной машине Java диапазоне (который может быть изменен администратором), что также будет защищать ОС и программиста от заведомо ложной реализации.

В случае одновременного доступа нескольких задач к одной и той же области данных в обоих языках существует система блокировок для синхронизации доступа. Относительно использования языков C/C++ и Java для написания многопоточных приложений можно отметить, что Java – менее ресурсоемкий язык и в нем управление памятью в большей степени автоматизировано и защищено от ошибочных действий программиста.

#### *Linda*

Базовыми языками для параллельных вычислений являются C/C++ и Fortran. При этом для распараллеливания приходится применять специализированные API, такие как OpenMP и MPI или встраивать операции организации межпроцессорного обмена данными из специальных языков, например, таких, как Linda.

Linda – язык программирования, предназначенный для параллельной обработки данных. Здесь параллельная программа – множество параллельных процессов, и каждый процесс работает согласно обычной последовательной программе. Все процессы имеют доступ к общей памяти, единицей хранения в которой является кортеж. Процессы работают с пространством кортежей по принципу: поместить кортеж, забрать, скопировать. В отличие от традиционной памяти, процесс может забрать кортеж из пространства кортежей, после чего данный кортеж станет недоступным остальным процессам. В отличие от традиционной памяти, изменить кортеж непосредственно в пространстве нельзя. В отличие от других систем программирования, процессы в системе Linda никогда не взаимодействуют друг с другом явно, и все общение всегда идет через пространство кортежей.

Для вычисления значения функции пользователя система Linda порождает параллельный процесс, на основе работы которого она формирует кортеж и помещает его в пространство кортежей. Параллельная

программа в системе Linda считается завершенной, если все порожденные процессы завершились или все они заблокированы функциями чтения/изменения.

#### *ADA*

ADA разработан специально для высоконадежных систем и включал в себя встроенную поддержку многозадачности, средства защитной блокировки, таймаутов, выполнения выборочного перестроения очереди клиентов, аварийного завершения. Механизмы межзадачного обмена данными и синхронизации основаны на концепциях рандеву и использовании защищенных объектов.

Основополагающая идея механизма рандеву достаточно проста. В спецификации задачи публикуются различные входы (entry) в задачу, в которых она готова ожидать обращения к ней от других задач. Далее, в теле задачи указываются инструкции принятия обращений к соответствующим входам, указанным в спецификации этой задачи. Поскольку задача-клиент и задача-сервер выполняются независимо одна от другой, то нет никакой гарантии, что обе задачи окажутся в точке осуществления рандеву одновременно. Поэтому, если задача-сервер оказалась в точке рандеву, но при этом нет ни одного обращения к входу (запроса на взаимодействие), то она должна ждать появления такого обращения. Аналогично, если задача-клиент обращается к входу, а задача-сервер не готова обслужить такое обращение, то задача-клиент должна ждать, пока задача-сервер обслужит это обращение. В процессе ожидания как задача-клиент, так и задача-сервер не занимают ресурсы процессора, находясь в состоянии, которое называют приостановленным или состоянием блокировки. Концепция защищенных объектов в ADA подобна мониторам: защищенные модули (типы и объекты) ADA инкапсулируют данные и позволяют осуществлять доступ к ним с помощью защищенных подпрограмм или защищенных входов. Стандарт языка гарантирует, что в результате выполнения кода таких подпрограмм и входов изменение содержимого данных будет производиться в режиме взаимного исключения без необходимости

создания дополнительной задачи. Однако защищенные объекты, по сравнению с мониторами, обладают большим преимуществом: протокол взаимодействия с защищенными объектами описывается барьерными условиями, а не низкоуровневыми и неструктурируемыми сигналами, используемыми в мониторах.

#### *LISP*

LISP создавался для обработки списков переменной длины и деревьев. Параллельность здесь основана на понятии отложенных действий: выполнимые действия в тексте замещаются на их результат, а невыполнимые преобразуются в остаточные, которые будут выполнены по мере появления дополнительной информации, таким образом, в LISP управление реализовано на основе потока данных. В последнее время разработано много расширений языка, одно из них — NewLISP. NewLISP - кроссплатформенный язык, доступный для Linux, Windows, MacOS X, OS/2. Он используется и в программировании для Интернет.

#### **Заключение**

Проведенный анализ представленных языков и архитектур вычислительных программ с точки зрения особенностей параллельного программирования позволяет выделить наиболее удобные для programмиста и наиболее ресурсосберегающие для ОС. Из всех языков можно отметить Java, C/C++ и Fortran с использованием специализированных API, таких как OpenMP и MPI. Однако с точки зрения создания параллельных приложений для кластера OpenMP не желательно использовать, так как в нем существует ограничение числа создаваемых потоков количеством доступных в системе процессоров, т.е. структура и быстродействие создаваемой программы напрямую будут зависеть от предоставленной аппаратной архитектуры.

С учетом того, что предложенная в [11] информационная технология реализуется на вычислительном кластере, то вполне вероятно, что узлы кластера могут представлять собой разные по мощности ЭВМ, нагрузка на которые будет неравномерно распределена. Это означает, что удобнее всего в качестве синхронизации процессов

здесь использовать барьерный метод либо монитор. Синхронизация данных возможна только с помощью обмена сообщениями. Исходя из этого и учитывая желаемую поддержку многопоточности, можно остановить выбор на двух средствах программирования: Java и API MPI.

### Список использованной литературы

1. Архитектуры и топологии многопроцессорных вычислительных систем [Электронный ресурс] /А.В. Богданов, В.В. Корхов, В.В. Мареев, Е.Н. Станкова. – ИНТУИТ.ру.– 2004. – 176с.– Режим доступа: <http://www.intuit.ru/department/hardware/atmcs>
2. Берилло А. NVIDIA CUDA – неграфические вычисления на графических процессорах [Электронный ресурс] /Берилло А.– 2008. – Режим доступа: <http://www.ixbt.com/video-3/cuda-1.shtml>
3. Воеводин В..В. Параллельная обработка данных: Курс лекций. [Электронный ресурс] / Воеводин В..В – Режим доступа к курсу: <http://parallel.ru/vvv/>
4. Гавва А. "Адское" программирование. Ada-95. Компилятор GNAT. [Электронный ресурс] /Гавва А. – 2004. – Режим доступа: <http://ada-ru.selfip.org:88/V-0.4w/index.html>
5. Гергель В.П. Технологии построения и использования кластерных систем. Интернет-университет информационных технологий / Гергель В.П. - ИНТУИТ.ру.: БИНОМ. Лаборатория знаний, 2007. – 512 с.
6. Городняя Л.В. Введение в программирование на Лиспе [Электронный ресурс] // Л.В. Городняя, Н.А. Березин. – Режим доступа: <http://www.intuit.ru/department/pl/lisp/>
7. Джоунз Г. Программирование на языке Оккам /Джоунз Г. – М.: Мир, – 1989. – 208 с.
8. Евсеев И. MPI [Электронный ресурс]: Вводный курс /Евсеев И. – Режим доступа: <http://www.ssd.sscc.ru/old/kraeva/MPI.html>
9. Инструменты разработчика NVIDIA CUDA. [Электронный ресурс]. – Режим доступа [http://www.nvidia.ru/object/tesla\\_software\\_ru.html](http://www.nvidia.ru/object/tesla_software_ru.html)
10. Ноутон П. Java 2. Наиболее полное руководство /П. Ноутон, Г. Шилдт.– С-Пб.: БНВ.– 2001.– 1072 с.
11. Паулин О.Н. Информационная технология распараллеливания решения нелинейных уравнений / О.Н. Паулин, Т.И. Усова// Тр. XI междунар. НПК "Современные информацион. и электрон. техн. "СИЭТ-2010". – Т.1. – Одесса: ДП "НЕПТУН-ТЕХНОЛОГИЯ", –2010. – С. 103.
12. Средства параллельного программирования в ОС Linux[Электронный ресурс]: Учеб. пособие /Под ред. Р.Х. Садыхова. – М.: ЕГУ, 2004. – 475 с. – Режим доступа: [http://www.opennet.ru/docs/RUS/linux\\_parallel/](http://www.opennet.ru/docs/RUS/linux_parallel/)
13. Чан Т. Системное программирование на C++ для Unix / Чан Т – К.: BHV,– 1997.– 592 с.

Получено 30.06.2010



Паулин  
Олег Николаевич,  
канд. техн. наук,  
профессор  
Одесск.нац.политехнич. ун-та paulin@te.net.ua



Усова  
Татьяна Ивановна,  
аспирант  
Одесск.нац.политехнич. ун-та usovati@rambler.ru