МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ДГТУ)

Кафедра «Математики и информатики»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
для проведения лабораторных занятий по дисциплине «Параллельное программирование»

Ростов-на-Дону
ДГТУ
2022

УДК 004.8(075.8)

Составитель: А.И. Сухинов

Методические указания для проведения лабораторных занятий по дисциплине «Параллельное программирование» – Ростов-на-Дону : Донской гос. техн. ун-т, 2022. – 21 с.

Методические указания содержат задания к лабораторным работам по параллельному программированию с использованием современных технологий организации параллельных вычислений на многопроцессорных вычислительных комплексах с распределенной или общей оперативной памятью.

Предназначены для обучающихся по направлению подготовки магистратуры 09.04.02 «Информационные системы и технологии».

УДК 004.8(075.8)

Печатается по решению редакционно-издательского совета
Донского государственного технического университета

Ответственный за выпуск зав. кафедрой «Математика и информатика» д-р физ.-мат. наук, профессор А.И. Сухинов

_____

В печать _____ г.
Формат 60×84/16. Объем _____усл. п. л.
Тираж _____экз. Заказ № _____

_____

Издательский центр ДГТУ
Адрес университета и полиграфического предприятия:
344000, г. Ростов-на-Дону, пл. Гагарина, 1

Лабораторная работа 1. Разработка простейших многопоточных программ

Цель работы: реализовать параллельные алгоритмы, учитывая особенности и правила многопоточности.

1. Разработать программу, реализующую параллельный алгоритм. Средства разработки: POSIX Threads, C++-threads.

2. Выполнить анализ эффективности параллельной программы: построить графики зависимости ускорения и коэффициента эффективности от числа потоков, оценить ускорение по закону Амдала (для нерекурсивных алгоритмов), оценить масштабирование.

Варианты заданий:

1. Генерация множества Мандельброта.

На вход программы поступают числа a, b, определяющие размер прямоугольной области для поиска точек, принадлежащих множеству Мандельброта. В результате выполнения программы необходимо получить подмножество n точек, принадлежащих множеству Мандельброта. Для этого прямоугольная область равномерно разделяется между потоками, и каждый поток выполняет поиск точек множества Мандельброта в своей области. Для проверки принадлежности точки множеству Мандельброта рекомендуется использовать алгоритм Escape Time. Необходимо учитывать проблему дисбаланса загрузки процессорных ядер и убедиться в корректности определения времени выполнения параллельной программы.

2. Алгоритм быстрой сортировки.

Входными данными для программы является неотсортированный числовой массив a, заполненный случайными значениями. Результат выполнения программы – отсортированный массив a. Необходимо реализовать рекурсивный алгоритм быстрой сортировки. При каждом вызове функции быстрой сортировки необходимо порождать поток. Порождение

потоков необходимо остановить при достижении количества потоков, равного заданному числу, которое варьируется от 1 до 10 числа процессорных ядер на вычислительном узле. В качестве опорного элемента выбирать первый элемент последовательности.

3. Алгоритм сортировки Шелла.

На вход алгоритма поступает неотсортированный массив a длины n, заполненный случайными числовыми значениями. Результат программы – отсортированный массив a. Длины di промежутков следует выбирать следующим образом: d1 = n / 2, d2 = n / 4, di = di – 1 / 2, dk = 1.

В ходе выполнения алгоритма сортировку каждого подмассива (для текущей длины d промежутка) необходимо выполнять в отдельном потоке. Число потоков не должно превышать количества процессорных ядер в системе.

4. Алгоритм бинарного поиска.

Необходимо реализовать две версии алгоритма поиска: для неотсортированного массива и для отсортированного массива. Ключи могут встречаться несколько раз в массиве. На вход подаётся случайный числовой массив a (отсортированный или неотсортированный), в котором осуществляется поиск, и набор чисел для поиска, представленных массивом b.

Выходные данные: если элемент массива b встречается в массиве a, то необходимо вывести индекс этого элемента в массиве a. В случае неотсортированного массива элементы массива a равномерно распределяется между потоками, каждый поток выполняет поиск элементов в своём подмассиве. В случае отсортированного массива целесообразно распределить между потоками числа для поиска, представленные массивом b. Число потоков не должно превышать количество процессорных ядер.

5. Решение системы линейных алгебраических уравнений (СЛАУ) методом Гаусса.

На вход программы поступает случайно сформированная СЛАУ Ax = b из n уравнений. Вид матрицы – произвольный. Выходные данные: вектор x

решений СЛАУ. При распараллеливании элементы матрицы равномерно распределяются между потоками по строкам. Необходимо реализовать параллельное выполнение как прямого, так и обратного хода метода Гаусса.

6. Алгоритм Прима построения минимального остовного дерева в графе.

Входные данные: случайный связный неориентированный граф G = (V, E). Для каждого ребра графа задана его стоимость. Выходные данные: набор вершин, образующих минимальное остовное дерево. На каждой итерации алгоритма Прима поиск ребра с наименьшей стоимостью выполняется параллельно. Для этого всё множество рёбер, инцидентных вершинам текущего остовного дерева, равномерно распределяется между потоками. Число потоков не должно превышать количество процессорных ядер.

7. Алгоритм Флойда поиска всех минимальных путей в графе.

Входные данные: случайный неориентированный граф G = (V, E) из вершин. Для каждого ребра графа задана его стоимость. Выходные данные: все минимальные пути в графе. На каждой итерации алгоритма множество всех вершин k, через которые может пройти кратчайший путь, равномерно распределяется между потоками. Число потоков не должно превышать количество процессорных ядер.

8. Алгоритм поиска подстрок в строке.

Входные данные: строка s, строка d. Выходные данные: номера символов, с которых начинается строка d, найденная в строке s. Элементы строки s равномерно распределяются между потоками, каждый поток выполняет поиск в своей подстроке. Число потоков не должно превышать количество процессорных ядер.

Контрольные вопросы:

1. В чём отличия потоков от процессов?

2. Какие ресурсы для потоков являются общими, какие для каждого потока уникальны?

3. Объяснить модель fork-join жизненного цикла потоков.

Проиллюстрировать создание, завершение, присоединение, отсоединение и принудительную отмену потоков на примере стандарта POSIX Threads.

4. Какие бывают политики планирования потоков? В чём особенности потоков реального времени?

Лабораторная работа 2. Потокобезопасные структуры данных

Цель работы: разработка и анализ масштабируемости потокобезопасной структуры данных.

1. Разработать программу, реализующую потокобезопасную структуру данных согласно варианту. Средства разработки: POSIX Threads, C++-threads.

2. Построить график зависимости пропускной способности структуры данных от числа потоков. Оценить масштабируемость.

Варианты заданий

1. Потокобезопасная односвязная очередь.

2. Потокобезопасная двусвязная очередь.

3. Потокобезопасный кольцевой буфер.

4. Потокобезопасная хеш-таблица с закрытой адресацией.

5. Потокобезопасное бинарное дерево поиска.

Замечания

Потокобезопасная структура данных должна поддерживать операции добавления, удаления и поиска элементов. Необходимо разработать тестовую программу для проверки работоспособности структуры данных. Алгоритм работы тестовой программы:

1. Создаётся p тестовых параллельных потоков.

2. Каждый поток выполняет n операций (вид операций выбирается случайным образом) для случайных элементов.

3. Измеряется время выполнений операций. Для корректного измерения времени необходимо синхронизировать потоки перед началом тестовых операций.

Контрольные вопросы:

1. Дать определение критической секции. Проиллюстрировать проблему взаимного исключения на примере выполнения несколькими потоками арифметической операции с глобальной переменной.

2. Дать определение мьютексу. Как реализован мьютекс? Какие основные операции он поддерживает?

3. В чём причины возникновения тупиковых ситуаций (deadlock)? Как можно их избежать?

Лабораторная работа 3. Редукция и префиксная сумма

Цель работы: научиться распараллеливать коллективные операции.

1.Написать программу, реализующую параллельный алгоритм выполнения заданной коллективной операции.

2. Построить графики ускорения и коэффициента эффективности параллельного алгоритма, оценить масштабируемость. Получить аналитическую формулу ускорения.

Варианты заданий

1. Редукции заданной ассоциативной операции $\otimes$ элементов массива.

2. Префиксная сумма для заданной ассоциативной операции $\otimes$ элементов массива.

Замечания:

1. Параметрами программы служат вид операции $\otimes$ и размер массива. Входной массив формируется случайным образом.

2. Необходимо провести серию экспериментов для массивов различной

длины. Длину массива рекомендуется выбирать максимально возможным, исходя из размера памяти вычислительного узла кластера.

Контрольные вопросы:

1. Почему необходимо проводить анализ эффективности параллельных программ? Как оценить эффективность параллельной программы?

2. Объяснить законы Амдала, Густавсона-Барсиса.

3. Из чего складывается время работы параллельной программы? Какими способами можно его минимизировать?

4. Что влияет на масштабирование параллельных программ?

Как увеличить масштабируемость?

Лабораторная работа 4. Паттерны многопоточного программирования

Цель работы: изучить и применить паттерны многопоточного программирования

1. Написать многопоточную программу согласно варианту задания.

2. Реализовать корректное завершение программы при поступлении сигналов. При поступлении сигнала SIGQUIT завершается обработка текущих операций и выполняется выход из программы. При поступлении сигналов SIGINT, SIGTERM выполняется аварийное завершение программы, при котором потоки принудительно отменяются.

3. Выполнить эксперименты, проанализировать эффективность параллельной программы. Выполнить оптимизацию параллельной программы.

Варианты заданий:

1. Модель потребитель-производитель (producer-consumer).

Реализовать модель потребитель-производитель. Для хранения данных использовать циклический буфер. Производитель генерирует случайные последовательности чисел и записывает каждую из них в отдельный файл. После этого имя файла добавляется в циклический буфер.

Потребитель ожидает появления в циклическом буфере элементов (последовательностей чисел). При поступлении новых элементов, потребитель удаляет задачу из буфера, считывает и сортирует последовательность и выводит отсортированную последовательность в тот же файл.

2. Конвейер (pipeline).

На вход программы через равные промежутки времени $t << 1$ с поступают имена текстовых файлов большого размера, которые помещаются в очередь на обслуживание.

В каждом файле необходимо выделить все слова, привести их к нижнему регистру, подсчитать статистику встречаемости слов, получить все уникальные слова, отсортировать их по алфавиту и вывести результаты (статистика встречаемости слов) в файл. Фазы алгоритма необходимо выполнять в отдельных потоках.

3. Очередь задач (work queue).

Входными данными является символьная строка s. Для строки s проверяется, есть ли в текущей директории файл с именем s. Если есть, выполняется попытка компиляции этого файла как программы на языке C; результат компиляции выводится в отдельный файл. Если файл успешно скомпилирован, скомпилированная программа запускается и результат её выполнения выводится в отдельный файл.

Если существует директория с именем s, тогда для каждого файла s' в этой директории решается аналогичная задача.

Каждый раз обработка директории выполняется в отдельном потоке. Необходимо обеспечить максимальный параллелизм при обработке строк.

4. Клиент-сервер (client-server).

Имеется файл, содержащий пары «ключ-значение» любого типа данных. Доступом к файлу обладает один поток-сервер. Потоки-клиенты через случайные промежутки времени $t << 1$ с отправляют запросы серверу. Сервер читает файл, выполняет команду клиента, при необходимости, записывает изменённый набор в файл, после чего отправляет результат выполнения клиенту. Запрос серверу состоит из команды и ключа, причём вид команды случайным образом выбирается из списка «найти», «добавить», «удалить», «выход».

Замечания:

Количество потоков во всех вариантах не должно превышать число процессорных ядер.

Для своего варианта задания самостоятельно выбрать подходящие показатели эффективности параллельной программы.

Контрольные вопросы:

1. С помощью каких примитивов синхронизации можно реализовать модель потребитель-производитель?

2. Дать рекомендации по использованию семафоров, условных переменных, барьеров. Какие недостатки имеют условные переменные? Почему функция ожидания для условной переменной (pthread_cond_wait) вызывается в цикле?

3. В чём преимущества циклического буфера перед массивом и линейным списком? Как можно реализовать циклический буфер, в чём преимущества каждого способа реализации?

4. Как можно обеспечить реентерабельность функций?

5. Объяснить необходимость использования локальных данных потоков (thread local storage).

6. В чём заключается сложность обработки сигналов в многопоточной среде? Как можно организовать обработку сигналов?

7. Почему необходимо обрабатывать принудительное завершение потоков?

Лабораторная работа 5. Многопоточное программирование на C++

Цель работы: изучить особенности многопоточного программирования на языке высокого уровня C++.

1. Разработать параллельную программу в соответствии с заданием.

Имеется n клиентов и m серверов (значение n близко к m). Каждый клиент и сервер характеризуются частотой λ и μ создания и обслуживания заявок соответственно.

Клиенты отправляют заявки в очередь (максимальная длина очереди q). Если в системе есть незанятый сервер, он снимает из очереди заявку и сообщает клиенту, отправившему эту заявку, значение среднего времени $t = 1/\mu$ обслуживания заявок данным сервером. После этого сервер приступает к выполнению заявки. Клиент, получив сообщение от сервера, ожидает время t, и если по истечении этого времени заявка не будет обслужена, то заявка считается невыполненной. Если сервер успел выполнить заявку за время $t' < t$, клиент формирует новую заявку (не дожидаясь окончания времени t).

Если для какого-либо сервера количество невыполненных заявок превысит количество выполненных более, чем на k, то этот сервер покидает систему. Если для какого-либо клиента количество выполненных заявок превысит число невыполненных более, чем на k, то этот клиент покидает систему.

2. Провести эксперименты, определить среднее время обслуживания заявки в системе и среднюю длину очереди.

Замечания

Для хранения заявок необходимо написать потокобезопасную очередь заявок с мелкозернистыми блокировками. Очередь должна быть безопасна относительно исключений.

Рекомендуемые параметры модели:

$n = \{10, 11, \ldots 20\}$, $m = \{10, 11, \ldots 20\}$, $q \in \{10, 20, \ldots, 100\}$,

$\lambda \in \{0.1, 0.2, \ldots, 1\}$, $\mu \in \{0.1, 0.2, \ldots, 1\}$, $k \in \{10, 11, \ldots 20\}$

Каждому клиенту или серверу соответствует отдельный поток (асинхронная задача).

Выполнение заявки моделируется временной задержкой.

Измерение временных интервалов и генерация случайных чисел реализуются с помощью стандартных средств C++.

Контрольные вопросы:

1. В чём особенности создания и завершения потоков в C++?

2. Объяснить принцип паттерна RAII (Resource Acquisition Is Initialization). Как можно использовать RAII для работы с потоками? Привести примеры RAII из стандартной библиотеки.

3. Какие виды блокировок существуют в C++?

4. Чем отличаются асинхронные задачи (std::async) от потоков (std::thread)? Дать рекомендации по использованию асинхронных задач и потоков.

5. Описать принцип работы будущих результатов. В чём преимущества использования будущих результатов для синхронизации перед условными переменными? Как можно организовать хранение будущего результата, разделяемого несколькими потоками?

6. Для какой цели используются неблокируемые будущие результаты?

7. Привести пример реализации универсальной потокобезопасной обёртки данных.

Лабораторная работа 6. Параллельное программирование в стандарте MPI

Цель работы: изучение стандарта интерфейса обмена данными в параллельном программировании.

В работе по стандартизации MPI участвовало около 60 человек из 40 организаций, преимущественно из США и Европы. В разработку MPI были вовлечены большинство основных поставщиков параллельных компьютеров, исследователи из университетов, правительственных лабораторий и промышленности.

Целью MPI является создание широко используемого стандарта для написания программ на основе передачи сообщений. Следовательно, интерфейс должен быть практичным, мобильным, эффективным и гибким стандартом для передачи сообщений.

MPI (message passing interface) - стандартизованная и переносимая система передачи сообщений (библиотека функций). Стандарт определяет синтаксис и семантику библиотечных функций, используемых при написании переносимых программ с передачей сообщений на языках Fortran 77, C и C++.

Другими словами, MPI — это программный инструментарий для обеспечения связи между ветвями параллельного приложения.

Задания:

1. Разработать MPI-программу, реализующую параллельный алгоритм построения множества Мандельброта.

2. Проанализировать эффективность параллельной программы: построить графики зависимости ускорения и коэффициента эффективности от числа потоков, оценить масштабируемость, построить асимптотические оценки вычислительной и коммуникационной сложности параллельного алгоритма.

Замечания

В качестве входных данных используется размер области a, b. Программа должна выдавать список точек из данной области, принадлежащих множеству Мандельброта.

Для проверки принадлежности точки множеству Мандельброта рекомендуется использовать алгоритм Escape Time.

В процессе разработки MPI-программы необходимо учитывать проблему дисбаланса загрузки процессорных ядер и убедиться в корректности определения времени выполнения параллельной программы.

Контрольные вопросы:

1. Привести архитектурные особенности распределённых вычислительных систем.
2. Дать определение стандарту MPI. В чём отличие программирования в стандарте MPI от многопоточного программирования?
3. Пояснить термины ранг и коммуникатор.
4. Как происходит запуск MPI-программы на вычислительном кластере?
5. В чём состоит принцип распараллеливания в стандарте MPI?

Лабораторная работа 7. Создание различных топологий в MPI

Цель работы: изучение функций создания декартовой топологии (решетки) и топологии графа в MPI.

Топология – механизм сопоставления процессам альтернативной схемы адресации. В MPI топологии виртуальны, не связаны с физической топологией сети.

Топология используется для более удобного обозначения процессов, и,

таким образом, приближения параллельной программы к структуре математического алгоритма.

В некоторых случаях топология может использоваться системой для оптимизации распределения процессов по физическим процессорам при помощи изменения порядка нумерации процессов внутри коммуникатора.

Два типа топологий:

– декартова (прямоугольная решётка произвольной размерности);

– топология графа.

Для создания коммуникатора с топологией типа граф в MPI предназначена функция:

```
int MPI_Graph_create(MPI_Comm oldcomm, int nnodes, int *index, int *edges,
 int reorder, MPI_Comm *graphcomm),
где:
 - oldcomm - исходный коммуникатор,
 - nnodes - количество вершин графа,
 - index - количество исходящих дуг для каждой вершины,
 - edges - последовательный список дуг графа,
 - reorder - параметр допустимости изменения нумерации процессов,
 - cartcomm – создаваемый коммуникатор с топологией типа граф.
```

Операция создания топологии является коллективной и, тем самым, должна выполняться всеми процессами исходного коммуникатора.

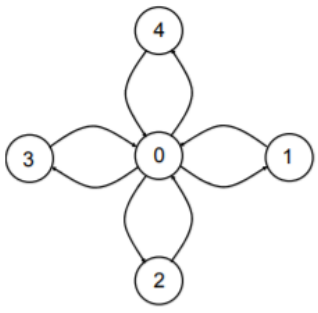

Рисунок 1 - Пример графа для топологии типа звезда

Для примера создадим топологию графа со структурой, представленной на рис1.

В этом случае количество процессов равно 5, порядки вершин

(количества исходящих дуг) принимают значения (4,1,1,1,1), а матрица инцидентности (номера вершин, для которых дуги являются входящими) имеет вид:

Процессы	Линии связи
0	1, 2, 3, 4
1	0
2	0
3	0
4	0

Для создания топологии с графом данного вида необходимо выполнить следующий программный код:

```
// создание топологии типа звезда
int index[] = { 4,1,1,1,1 };
int edges[] = { 1,2,3,4,0,0,0,0 };
MPI_Comm StarComm;
MPI_Graph_create(MPI_COMM_WORLD, 5, index, edges, 1, &StarComm);
```

Стоит отметить две полезные функции для работы с топологиями графа.

Количество соседних процессов, в которых от проверяемого процесса есть выходящие дуги, может быть получено при помощи функции:

```
int MPI_Graph_neighbors_count(MPI_Comm comm,int rank, int *nneighbors).
```

Получение рангов соседних вершин обеспечивается функцией:

```
int MPI_Graph_neighbors(MPI_Comm comm,int rank,int mneighbors, int *neighbors),
```

где mneighbors есть размер массива neighbors.

Задание:

1. Реализовать декартову топологию.

Для создания декартовой топологии (решетки) в MPI предназначена функция:

```
int MPI_Cart_create(MPI_Comm oldcomm, int ndims, int *dims, int *periods,
 int reorder, MPI_Comm *cartcomm),
```

где:

- oldcomm - исходный коммуникатор,
- ndims - размерность декартовой решетки,
- dims - массив длины ndims, задает количество процессов в каждом

измерении решетки,

- periods - массив длины ndims, определяет, является ли решетка периодической вдоль каждого измерения,

- reorder - параметр допустимости изменения нумерации процессов,

- cartcomm – создаваемый коммуникатор с декартовой топологией процессов.

Контрольные вопросы:

1. Как определить время выполнения MPI программы?

2. В чем различие парных и коллективных операций передачи данных?

3. Какая функция MPI обеспечивает передачу данных от одного процесса всем процессам?

4. Что понимается под производным типом данных в MPI?

5. Какие способы конструирования типов имеются в MPI?

Список использованных источников

1. Антонов А.С., Воеводин Вл.В. Эффективная адаптация последовательных программ для современных векторно-конвейерных и массивно- параллельных супер-ЭВМ. // Программирование. 1996, 4, с. 37-51.
2. Афанасьев К.Е. Многопроцессорные вычислительные системы и параллельное программирование: Учебное пособие/ Афанасьев К.Е., Стуколов С.В., Демидов А.В., Малышенко В.В.; Кемеровский госуниверситет. - Кемерово: Кузбассвузиздат, 2003. - 182 с.
3. Бандман О.Л. Клеточно-нейронные модели пространственно-временной динамики. // Программирование. 1999, N 1, с. 4-17.
4. Бахвалов Н.С. Численные методы // Главня редакция физико-математической литературы изд-ва "Наука", М., 1975 г. - 632 с.
5. Валиев М.К. Применение временной логики к спецификации программ. // Программирование. 1998, 2, с. 3-9.
6. Валиев М.К. Применение временной логики к спецификации программ. // Программирование. 1998, 2, с. 3-9.
7. Воеводин В.В. Математические модели и методы в параллельных процессах. - М.: Наука, 1986. - 296 с.
8. Воеводин В.В. Просто ли получить обещанный гигафлоп? // Программирование. 1995, N 4, С. 13-23.
9. Воеводин В.В. Суперкомпьютеры: вчера, сегодня, завтра. // Сборник научно-популярных статей Российская наука на заре нового века. Под редакцией академика В.П. Скулачева. М.: научный мир, 2001. С. 475-483.
10. Галатенко В.А., Костюхин К.А. Отладка и мониторинг распределенных разнородных систем. // Программирование, 2002, 1. С. 27-37.
11. Гергель В.П., Стронгин Р.Г. Основы параллельных вычислений для многопроцессорных вычислительных систем. Учебное пособие. Нижний Новгород: Изд-во ННГУ им. Н.И. Лобачевского, 2000. - 176 с.

12. Годунов С.К., Рябенький В.С. Разностные схемы. - М.: Наука, 1973. - 400 с.

13. Дацюк В.Н., Букатов А.А., Жегуло А.И. Электронное методическое пособие по курсу "Многопроцессорные системы и параллельное программирование" Часть I. Введение в организацию и методы программирования многопроцессорных вычислительных систем. Ростов-на-Дону, 2000.

14. Демидов А.В., Сидельников К.В. Эмуляция параллельной обработки данных на персональном компьютере // XLI Международная научная студенческая конференция "Студент и научно-технический прогресс". Сб. трудов. Новосибирск, 2003. С. 110-111.

15. Дмитриева О.А. Параллельные алгоритмы численного решения систем обыкновенных дифференциальных уравнений. //Мат. Моделирование N 5, 2000, С. 81-86.

16. Дуйсекулов А.Е., Елизарова Т.Г. Использование многопроцессорных вычислительных систем для реализации кинетически-согласованных разностных схем газовой динамики. // Математическое моделирование, 1990, т. 2, N 7, С. 139-147.

17. Ершов Н.М. Построение графов вычислительных алгоритмов методом автотрассировки. // Программирование. 2000, N 6, с. 58-64.

18. Заворин А.Н. Параллельное решение линейных систем при моделировании электрических цепей. // Математическое моделирование, 1991, т. 3, N 3, С. 91-96.

19. Захарьева Н.Л., Хозиев В.Б., Ширков П.Д. Моделирование и образование.// Математическое моделирование, 1999, т. 11, N 5, с. 101-116.

20. Иванников В.П., Ковалевский Н.С., Метельский В.М. О минимальном времени реализации распределенных конкурирующих процессов в синхронных режимах. // Программирование. 2000, N 5, с. 44-52.

21. Ки-Чанг Ким. Мелкозернистое распараллеливание неполных гнезд циклов. // Программирование. 1997, N 2, С. 52-66.

22. Корнеев В.В. Параллельные вычислительные системы. М:"Нолидж", 1999. - 320 с.

23. Костенко В.А. К вопросу об оценке оптимальной степени параллелизма. // Программирование. 1995, 4, с. 24-28.

24. Крюков В.А. Операционные системы распределенных вычислительных систем. (учебный курс).

25. Крюков В.А., Удовиченко Р.В. "Отладка DVM программ" Программирование. - 2001. N. 3.-С.19-29.

26. Лабутин Д.Ю. Система удаленного доступа к вычислительному кластеру (менеджер доступа): Высокопроизводительные параллельные вычисления на кластерных системах. Материалы второго международного научно-практического семинара, Нижний Новгород: Издательство Нижегородского университета, 2002. С.184-187.

27. Лацис А.О. Как построить и использовать суперкомпьютер. М.: изд-во Бестселлер, 2003. 274 с.

28. Марков Н.Г. Мирошниченко Е.А., Сарайкин А.В. Моделирование параллельного программного обеспечения с использованием PS-сетей. // Программирование. 1995, N 5, с. 24-32.

29. Морозов В.А., Важенин А.П. Матричная арифметика многократной точности для параллельных систем с передачей сообщений. // Программирование. 1999, N1, С. 66-77.

30. Москвин Д.Б., Павлов В.А. Опыт использования MPI технологии для решения системы интегральных уравнений Фредгольма второго порядка. //Математическое моделирование, 2000, N 8, С. 3-8.

31. Немнюгин С.А., Стесик О.Л. Параллельное программирование для многопроцессорных вычислительных систем. - СПб.: БХВ-Петербург, 2002. - 400с.

32. Неупокоев Е.В., Тарнавский Г.А., Вшивков В.А. Распараллеливание алгоритмов прогонки: целевые вычислительные эксперименты. // Автометрия, N 4, том 38, 2002, стр. 74-87.

33. Салливан Э. Время — деньги. Создание команды разработчиков программного обеспечения/Пер, с англ. - М.: Издательско-торговый дом "Русская Редакция", 2002. - 368 стр.: ил.

34. Самарский А.А., Николаев Е.С. Методы решения сеточных уравнений. М.:Наука,1978. 561с.

35. Самофалов В.В., Коновалов А.В. Технология отладки программ для машин с массовым параллелизмом // "Вопросы атомной науки и техники". Сер. Математическое моделирование физических процессов. 1996. Вып. 4. С. 52-56.

36. Самофалов В.В., Коновалов А.В., Шарф С.В. Динамизм или статичность: поиск компромисса // Труды Всероссийской научной конференции "Высокопроизводительные вычисления и их приложения". М., 2000. С. 165-167.

37. Сапожников А.П., Сапожникова Т.Ф. Реинженирингговая технология распределенных вычислений в локальной сети. Труды международной конференции "Распределенные вычисления и Грид-технологии в науке и образовании" (Дубна, 29 июня-2 июля 2004 г.). 11-2004-205, Дубна, ОИЯИ, 2004. Стр.183-190.

38. Тарнавский Г.А., Шпак С.И. Декомпозиция методов и распараллеливание алгоритмов решения задач аэродинамики и физической газовой динамики: вычислительная система "Поток-3". // Программирование. 2000, N 6, С. 45-57.

39. Шалыто А.А. Автоматное проектирование программ. Алгоритмизация и программирование задач логического управления. Журнал "Известия академии наук. Теория и системы управления" Номер 6. Ноябрь-Декабрь 2000. С.63-81.

40. Шпаковский Г.И. Архитектура параллельных ЭВМ. - Минск, 1989. - 136 с.