ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

МОСКОВСКИЙ ИНЖЕНЕРНО-ФИЗИЧЕСКИЙ ИНСТИТУТ
(ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ)

А.Б. ВАВРЕНЮК, В.В. МАКАРОВ, Е.В. ЧЕПИН

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ПАРАЛЛЕЛЬНОМ СИ ДЛЯ МРР-СИСТЕМ

Лабораторный практикум

Под редакцией Е.В. Чепина

Учебное электронное издание

Москва 2008

УДК 004.4(076.5)
ББК 32.973-18я7
В12

Вавренюк А.А., Макаров В.В., Чепин Е.В. **Основы программирования на Параллельном Си для МРР-систем.** *Лабораторный практикум* / Под ред. Е.В. Чепина [Электронный ресурс]. – М.: МИФИ, 2008. – 76 с.

Предназначен для изучения основ архитектуры и технологии параллельного программирования широко распространенных МРР-систем, построенных на базе транспьютеров TMS320C40 или аналогичных. Кроме справочного и методического материала описывается методика и предлагаются варианты индивидуальных заданий для выполнения двух лабораторных работ на 4-процессорной системе, используемой в учебном процессе кафедры «Компьютерные системы и технологии» МИФИ уже около десяти лет.

Рекомендуется использовать при изучении дисциплин «Технологии программирования. Параллельное программирование», «Архитектура вычислительных систем», «Обработка сигналов и изображений» для студентов групп К5-12в, К5-12с, К9-121, 122, 123, 124, 125, 126 факультета «Кибернетика» и групп В5-12, В5-12п вечернего факультета, обучающихся по специальности «Вычислительные машины, комплексы, системы и сети». Также будет полезно всем студентам, изучающим программирование для многопроцессорных вычислительных систем.

Рецензент канд. техн. наук, доц. А.А. Храмов

Рекомендовано к изданию редсоветом МИФИ

ISBN 978-5-7262-0944-9

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ....4
ВВЕДЕНИЕ....5
1. АППАРАТНЫЕ СРЕДСТВА....7
1.1. Сигнальные процессоры TMS320C4x....7
1.2. Краткое описание TIM SMT302....11
1.3. Материнская плата SMT 320....12
2. РАЗРАБОТКА ПРОГРАММ НА ЯЗЫКЕ ПАРАЛЛЕЛЬНЫЙ СИ....15
2.1. Языковые средства параллельного программирования на TMS320C4015
2.2. Язык Параллельный Си....19
2.3. Язык конфигурации....24
2.4. Пример создания приложения и конфигурационного файла....30
2.5. Технология работы в дисплейном классе....31
2.6. Методические указания по отладке и выполнению параллельной программы....34
3. ОПИСАНИЯ ЛАБОРАТОРНЫХ РАБОТ....37
3.1. Лабораторная работа 1. Знакомство с многопроцессорной системой TMS320C40 (вводная работа)....37
3.1.1. Порядок выполнения работы....37
3.1.2. Пример задания....38
3.1.3. Варианты заданий к лабораторной работе 1....41
3.2. Лабораторная работа 2. Изучение технологии таймирования параллельной программы....52
3.2.1. Способы временных оценок параллельных программ....52
3.2.2. Рекомендуемые топологии....55
3.2.3. О векторной библиотеке Veclib....56
3.2.4. Методические указания к лабораторной работе 2....57
3.2.5. Задание по лабораторной работе 2....61
3.2.6. Варианты заданий к лабораторной работе 2....62
СПИСОК ЛИТЕРАТУРЫ....68
Приложение 1. Пример программы на языке Параллельный Си: интегрирование методом Симпсона на 1 процессоре с использованием библиотеки Veclib и технологии таймирования....69
Приложение 2. Библиотека некоторых функций Параллельного Си....71

ПРЕДИСЛОВИЕ

Цель данного практикума – изучение основ архитектуры и технологии параллельного программирования MPP-систем, построенных на базе TMS320C40 (или ему подобных). Изучение предполагает выполнение студентом индивидуальных лабораторных работ на 4-процессорной системе, в ходе которых приобретаются начальные практические навыки разработки на языке Параллельный Си, отладки и исследования эффективности функционирования параллельных программ.

Разработчиком клиент-серверного программного обеспечения (ПО) для поддержки данного лабораторного практикума является доцент А.Б. Вавренюк. Большую помощь в разработке и отладке ПО оказал выпускник кафедры «Компьютерные системы и технологии» МИФИ Г.В. Шестоперов.

Материал лабораторной работы 1 (разд. 3.1) подготовили доценты А.Б. Вавренюк и В.В. Макаров. Ими также написаны разд. 2.4, 2.5, 2.6.

Остальной материал, который связан с аппаратно-программной частью, лабораторную работу 2 (разд. 3.2), а также общее редактирование выполнил доцент Е.В. Чепин.

Авторы данного пособия выражают большую благодарность выпускнику кафедры «Компьютерные системы и технологии» МИФИ А.Н. Никитину за аппаратную инсталляцию и тестирование сервера с материнской платой SMT 320 с четырьмя процессорами TMS320C40.

ВВЕДЕНИЕ

В настоящее время решение ряда научных и практических задач (в области физики, химии, биологии, искусственного интеллекта, управления сложными объектами в реальном масштабе времени и др.) требует создания новых средств обработки информации, удовлетворяющих требованиям сверхвысокой производительности, а также повышенной надежности и живучести.

Компьютеры с архитектурой Дж. фон Неймана не могут обеспечить выполнения требований таких задач по производительности из-за присущих последовательным однопроцессорным машинам ограничений на быстродействие, с одной стороны, и в силу внутреннего параллелизма алгоритмов ряда прикладных областей, с другой стороны. Одним из основных подходов к решению этих проблем является использование методов и средств параллельной обработки информации, развиваемых в области алгоритмов решения задач, технических средств и средств программирования. Внутренний параллелизм некоторой задачи, согласно классификации Треливена (P.C. Treleaven), разумно реализовывать в зависимости от принадлежности с помощью различных архитектурных принципов (табл. В.1).

Таблица В.1

Уровень параллелизма	Архитектура
Независимые задания, шаги задания и программы, подпрограммы	Мультипроцессорные системы
Циклические структуры и итерации	Векторные процессоры
Операторы и команды	Многофункциональная (суперскалярная) обработка
Внутренний параллелизм на уровне команды (фазы команд)	Конвейерная архитектура

Разработка концепций параллелизма в области средств вычислительной техники привела к созданию многомашинных и многопроцессорных (мультипроцессорных) вычислительных систем (МВС).

Эффективная реализация высокопроизводительных МВС возможна лишь при использовании современной элементной базы на основе СБИС. Появление в 80-х годах в Великобритании СБИС названного «транспьютер» (см., например, [1 – 4]), специализированного процессора для простого построения мультипроцессорных систем большой размерности, дало мощный толчок в развитии концепций и технологии построения как аппаратно-программных средств (фирмы Inmos, Transtech, Parsytech и др.), так и параллельного программирования (в частности, фирма 3L) для мультипроцессорных MPP-систем (massively parallel processing), например [1, 2, 4, 5]. Транспьютер – термин, описывающий семейство программируемых СБИС и обладающих следующими основными параметрами и свойствами:

- 16- или 32-разрядные RISC-процессоры общего назначения;
- аппаратно-поддерживающие операции с плавающей точкой;
- имеющие четыре канала обмена (линка), которые напрямую соединяются с линками других, себе подобных процессоров.

Развитие этих принципов привело во второй половине 80-х годов к созданию технологии MPP (соответствующий русский термин – системы с массовым параллелизмом), включающей в себя комплекс аппаратно-программных средств. Эта технология является в последние два десятилетия одной из основных архитектур для создания современных суперкомпьютеров с производительностью 10^{11} – 10^{15} Mflops. Современные большие MPP-системы состоят из многих тысяч, и даже сотен тысяч процессоров, обеспечивающих эффективное распараллеливание вычислений, однородность и регулярность структур вычислительных систем (ВС). В качестве процессорного элемента в них используются современные RISC-процессоры, в том числе в ряде реализованных в 90-е годы прошлого столетия проектов, применялся TMS320C40 фирмы Texas Instruments, который во многом является преемником транспьютера [1].

Появление MPP-систем потребовало создания для них новых программных средств, обеспечивающих, с одной стороны, эффективную организацию и управление параллельными вычислениями (системное программное обеспечение), а с другой стороны – технологию разработки прикладных программ в виде совокупности параллельно выполняемых задач/программ.

1. АППАРАТНЫЕ СРЕДСТВА

1.1. Сигнальные процессоры TMS320C4x

Семейство процессоров TMS320C40 (рис. 1.1) фирмы Texas Instruments разработано специально для систем параллельных вычислений и вычислений в реальном времени, реализующих парадигму MPP-систем (massively parallel processing). Эти процессоры обеспечивают производительность 330 Mops или 60 Mflops и пропускную способность 380 Мбайт/с. Особое внимание уделено обеспечению быстрых межпроцессорных связей с одновременным обменом по всем шести каналам ввода-вывода через коммуникационные порты (линки).

Коммуникационные порты (линки) служат для освобождения шин памяти от межпроцессорного обмена и обеспечивают пропускную способность 120 Мбайт/с (6 линков по 20 Мбайт/с каждый). Без них пропускная способность внешних интерфейсов памяти была бы уменьшена, что снизило бы и производительность процессора.

Процессор прямого доступа к памяти (DMA Coprocessor) освобождает CPU от передачи данных внутри процессорной памяти. При этом вся мощность CPU (240 Mops) фокусируется только на вычислительных задачах, в то время как DMA-процессор использует его 90 Mops для ввода-вывода.

CPU TMS320C40 имеет регистровую архитектуру и включает в себя:

умножитель целых/с плавающей точкой чисел – умножение за один цикл 32-битных целых или 40-битных чисел с плавающей точкой;

АЛУ целых/с плавающей точкой чисел – арифметические операции за один цикл 32-битных целых или 40-битных чисел с плавающей точкой;

32-битный циклический сдвигатель;

два внешних регистровых АЛУ (вычисляют два адреса за цикл);

файл регистров CPU – 32 регистра.

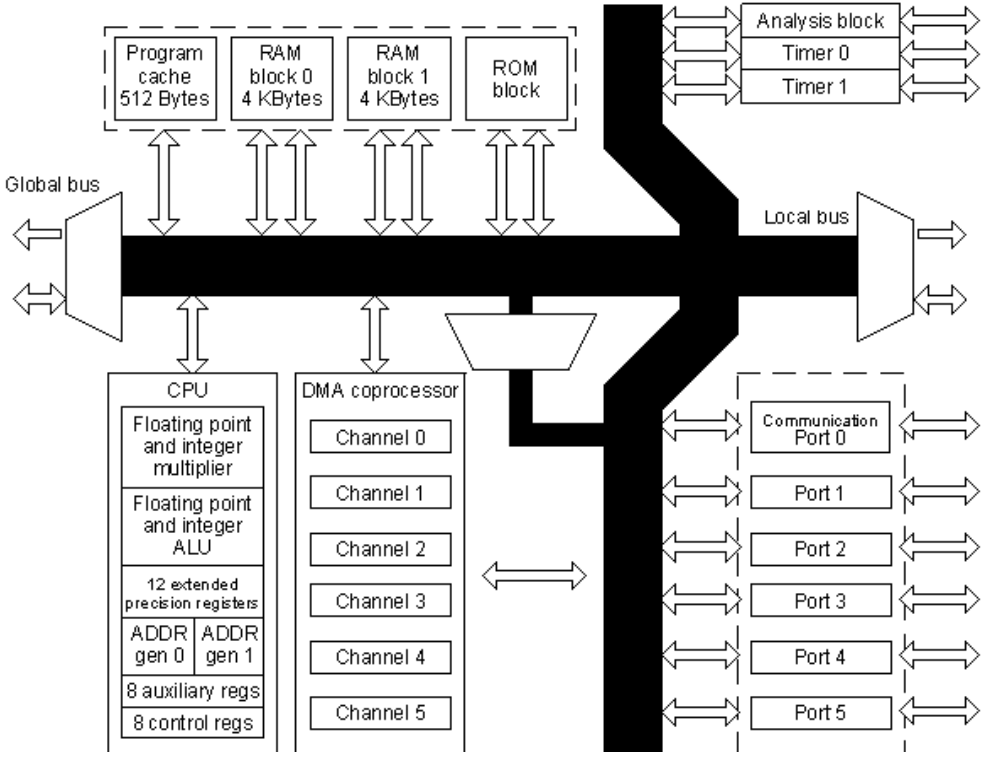

Рис. 1.1. Архитектура процессора TMS320C40

Для достижения полной производительности TMS320C40 была разработана специальная архитектура памяти и шин. TMS320C40 может передавать до четырех 32-битных слов за один цикл: программный код, два операнда и данные DMA-процессора. Внутренние шины могут передавать все четыре слова параллельно, выбирая из семи источников данных.

Процессор TMS320C40 имеет два внутренних блока статической памяти SRAM (1K x 32-bits каждый) и внутреннее постоянное запоминающее устройство (ПЗУ). Каждый блок обеспечивает возможность двух обращений за цикл. TMS320C40 также включает в себя 128x32-bit кэш-команд, который используется для хранения часто используемых участков кода, что резко уменьшает количество обращений к более медленной внешней памяти.

Первичный регистровый файл центрального процессорного устройства (ЦПУ) представляет собой многовходовой файл из 32 регистров. Все регистры первичного регистрового файла могут использоваться умножителем, арифметико-логическим устройством (АЛУ) и в качестве регистров общего назначения. Регистры имеют некоторые специальные функции. Например, 12 регистров повышенной точности могут использоваться для размещения результатов операций с плавающей точкой, восемь дополнительных регистров – для некоторых косвенных способов адресации, а также как целочисленные и логические регистры общего назначения. Остальные регистры обеспечивают такие функции системы, как адресация, управление стеком, прерывания, отображение статуса процессора, повторы блоков команд.

Регистры повышенной точности предназначены для хранения и обработки 32-разрядных целых чисел и 40-разрядных чисел с плавающей точкой. Дополнительные регистры доступны как для АЛУ, так и для двух модулей адресной арифметики. Основная функция этих регистров – генерация 32-разрядных адресов. Они также могут использоваться как счетчики циклов или как регистры общего назначения.

Краткая структурная нотация [5, 6] процессора TMS320C40 приведена ниже:

$$P(\text{TMS320C40})_{\text{RISC}} = \mathbf{I_p}^{\mathbf{20}}{}_{\mathbf{32,40}}\,[E,\ M(\text{внутр}),\ PDMA,\ 6\text{Link}^{\,20\text{MB/s}}]$$

$$\mathbf{E} = \{B_{\mathbf{p32}}(\text{АЛУ}),\ F_{p40}(\text{АЛУ}),\ B_{p32}(*),\ F_{p40}(*),\ 32Rg_{32}\}$$

$$M(\text{внутр}) = \{\text{Cashc}_{\ 128x32},\ 2\text{SRAM}_{\ 1Kx32},\ \text{ROM}\}$$

TMS320C40 имеет две внешних шины памяти, называемых локальной и глобальной. Они адресуются 32-битным адресом и отличаются значением верхнего бита. TMS320C40 может иметь одновременный доступ на обе шины. Обмен на этих шинах производится 32-битными словами. На рис. 1.2 схематично показаны основные внешние выводы процессора. К ним относятся: выводы локальной (Local bus) и глобальной (Global bus) шины, шесть коммуникационных портов (линков), выводы прерываний.

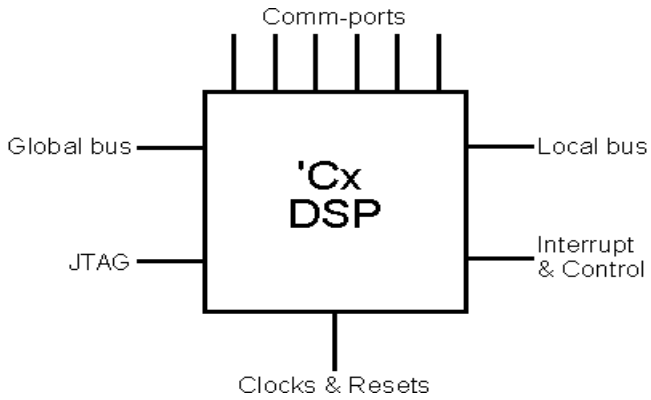

Рис. 1.2. Схема основных внешних выводов процессора TMS320C40: Comm-ports – последовательные порты; Global bus – глобальная шина; Local bus – локальная шина; JTAG – порт; Interrupt and Control – прерывания и контроль; Clocks and Resets – синхронизация и сброс

В настоящее время существует несколько фирм, производящих аппаратное обеспечение для параллельных вычислений на основе семейства сигнальных процессоров TMS320C4x. Типовой набор оборудования такой системы обычно состоит из нескольких TIM-модулей, содержащих, как правило, один процессор TMS320C4x, память DRAM, разъемы для подключения к материнской плате. Материнская плата для TIM-модулей выполнена в виде платы расширения IBM PC с интерфейсом ISA или PCI для подключения к HOST-ЭВМ. На плате обычно может быть установлено до 4 TIM. Имеются также разъемы для соединения процессоров между собой, некоторые из таких связей жестко заданы и не могут быть изменены. Материнская плата содержит также устройства, дающие

возможность передавать информацию в/из HOST-ЭВМ, отлаживать программы и др. В свободные слоты HOST-ЭВМ может быть установлено несколько материнских плат.

Пользователь-программист использует HOST-ЭВМ для создания, отладки и выполнения параллельной программы. На HOST-ЭВМ транслируются программы для каждого из процессоров, формируется загрузочный модуль приложения для всей сети процессоров, производится запись кода в память процессоров, запуск программы, управление запросами по обмену данными между сетью процессоров и памятью HOST-ЭВМ и вывод результатов программы.

1.2. Краткое описание TIM SMT302

Конструктивно процессор устанавливается на небольшой плате, которая в нашем случае маркирована как TIM SMT302 (Typical Interchange Module – типовой элемент замены для материнской платы SMT 320 (Stand Module Transputer)). Схема TIM-модуля показана на рис. 1.3.

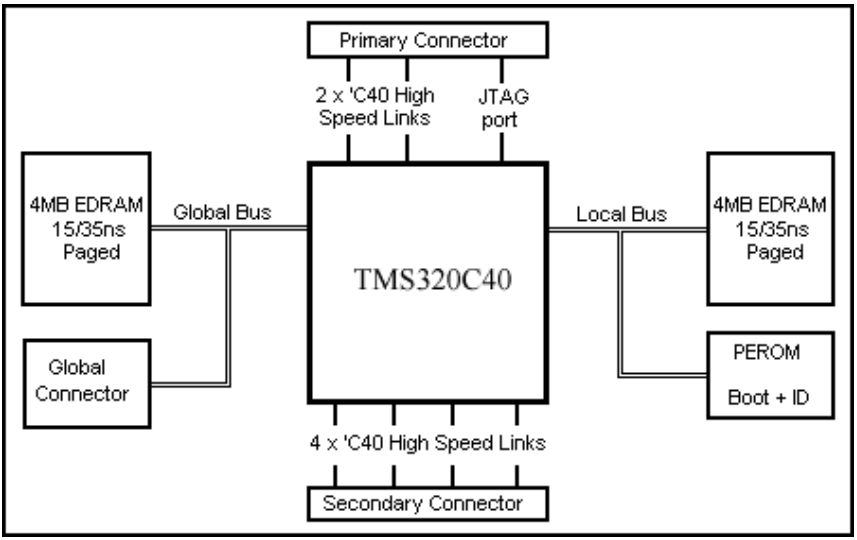

Рис. 1.3. Схема подключения основных внешних блоков TIM SMT302:
Global Connector – глобальный разъем; Primary Connector – первый разъем;
Secondary Connector – второй разъем; High Speed Links – высокоскоростные линки (каналы связи)

В состав TIM SMT302 входят:

TMS320C40 (50 или 60 МГц) параллельный сигнальный процессор;

8 Мбайте EDRAM;

шесть 20 Мбайт/с межпроцессорных коммуникационных портов (линков) шириной 1 байт;

быстрый интерфейс памяти;

разъем для подключения к глобальной шине;

32 Кбайт ПЗУ для загрузки и самотестирования;

JTAG-порт диагностики и отладки.

1.3. Материнская плата SMT 320

Основным конструктивным элементом мультипроцессорной системы является материнская плата, которая вставляется в разъем расширения (обычно PCI-слот) материнской платы персонального компьютера или сервера, который будет далее называться HOST-ЭВМ, или просто HOST. На рис. 1.4 изображена схема материнской платы SMT 320.

В состав материнской платы SMT302 входят:

четыре гнезда для TIM;

интерфейс PCI для связи с HOST-ЭВМ, связанный с линком 3 первого TIM (root), обеспечивающий скорость обмена до 10 Мбайт/с;

разъем глобальной шины на слоте 0 для связи с HOST-ЭВМ, обеспечивающий скорость обмена до 50 Мбайт/с.

Все TIM связаны в кольцо через линки 2 и 5 (рис. 1.5). Остальные линки могут быть физически соединены в произвольном порядке через внешние кабели. Линки также могут быть связаны с процессорами на других материнских платах, расположенных не дальше 50 см. Все линки не буферизированы и обеспечивают скорость обмена 16 Мбайт/с.

Драйвер для ОС Windows 98 обеспечивает программный интерфейс, позволяющий инициировать передачу блоков в/из памяти HOST-ЭВМ, чтение-запись в управляющие регистры SMT 320.

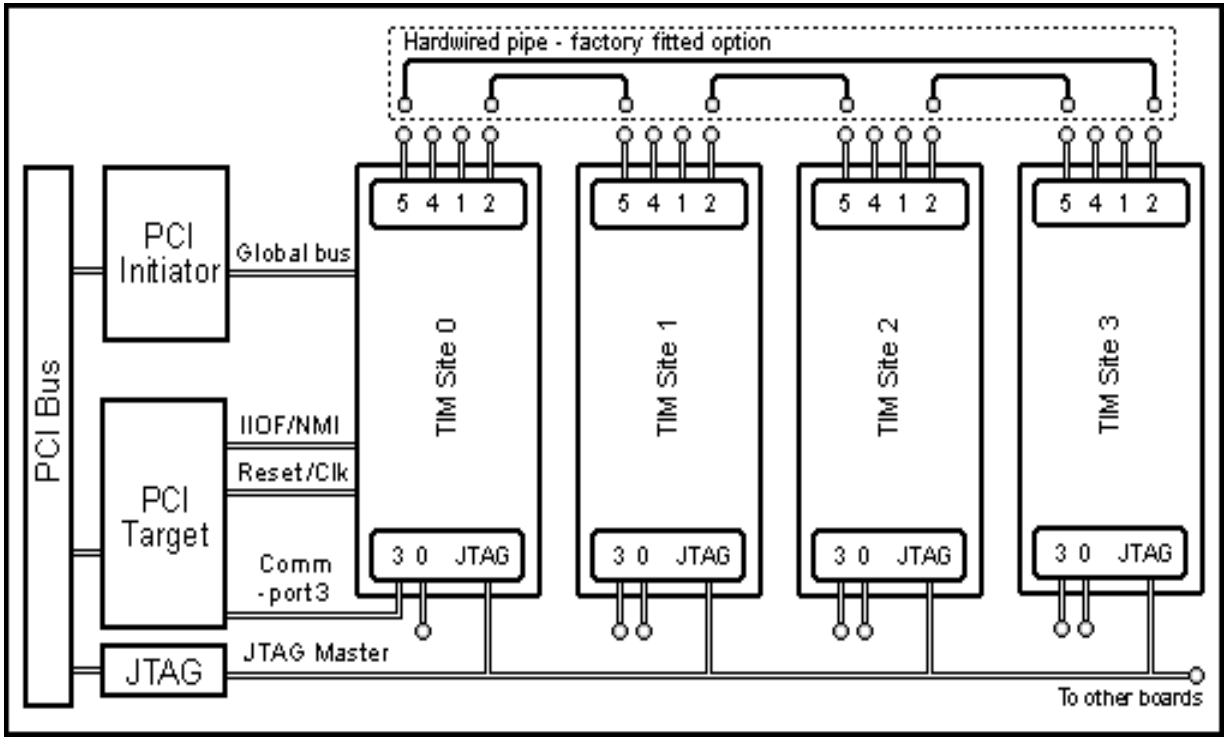

Рис. 1.4. Схема материнской платы SMT 320

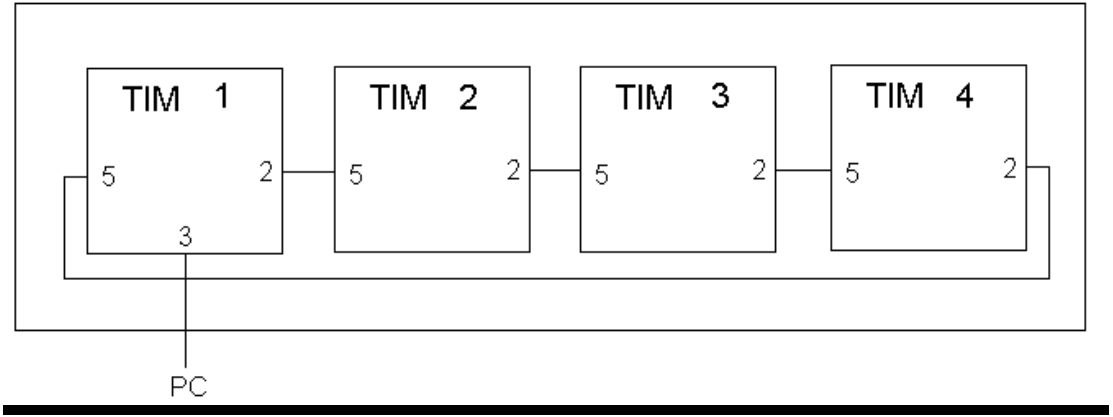

Рис. 1.5. Схема материнской платы SMT 320

2. РАЗРАБОТКА ПРОГРАММ НА ЯЗЫКЕ ПАРАЛЛЕЛЬНЫЙ СИ

Система программирования Параллельный Си содержит интегрированный набор языковых и программных средств, необходимых для разработки и отладки как обычных последовательных, так и параллельных программ, исполняемых в процессорной сети. В Параллельный Си прикладная программа является совокупностью конкурентно исполняемых задач, взаимодействующих через каналы. В общем случае задача как программный модуль состоит из последовательности операторов. Возможны задачи с внутренним параллелизмом в виде нескольких конкурентно исполняемых ветвей (thread). Каждая ветвь (thread) состоит из последовательности операторов. Однако в рамках данного практикума эта технология не изучается.

Для разработки параллельных программ необходимо владение навыками последовательного и параллельного программирования. Техника последовательного программирования на языке Си в достаточной степени изложена в литературе.

2.1. Языковые средства параллельного программирования на TMS320C40

Появление многопроцессорных вычислительных систем (МВС) привело к необходимости создания языков и систем программирования, позволяющих описывать параллельные процессы и эффективно их реализовывать. Языки параллельного программирования (ЯПП) предназначены для описания алгоритмов решения сложных задач в виде ансамбля взаимодействующих процессов [2].

Основным понятием ЯПП является процесс. Процессы в ЯПП могут быть элементарные и составные. Элементарные процессы могут состоять из одного или совокупности последовательно выполняемых операторов в виде блоков, подпрограмм, процедур,

функций, задач. Объем элементарных процессов определяет степень грануляции или «зернистости» параллельной программы. Программа в ЯПП является системой процессов, выполняемых конкурентно – путем разделения одного или нескольких процессоров, либо параллельно – путем выделения каждому процессу индивидуального процессора.

В первом случае осуществляется мультипрограммирование, оно обычно поддерживается ядром операционной системы.

Во втором случае осуществляется мультипроцессорная обработка (для процессоров с общей памятью) или распределенная обработка (для процессоров, связанных коммуникационными каналами).

Для обеспечения совместной работы при решении сложной задачи параллельно выполняемые процессы должны обмениваться данными и синхронизироваться.

Синхронизация обеспечивает некоторый предписываемый порядок взаимодействия процессов во времени. Синхронизация необходима в двух случаях:

1) когда определенное действие одного процесса должно быть выполнено только после завершения определенного действия другого процесса;

2) когда необходимо обеспечить заданную дисциплину доступа к разделяемым ресурсам.

Для ЯПП разработаны различные модели и механизмы взаимодействия процессов. Основными моделями взаимодействия процессов в ЯПП являются:

- взаимодействия через общие (разделяемые) нелокальные переменные;
- взаимодействия посредством передачи сообщений.

Хотя эти модели являются машинно-независимыми, но на стадии реализации из соображений эффективности приходится учитывать архитектурные особенности используемых ВС.

В данном лабораторном практикуме будем знакомиться с технологией параллельного программирования, основанной на взаимодействии процессов посредством передачи сообщений.

В моделях взаимодействия процессов на основе передачи сообщений отсутствуют разделяемые переменные, данные передаются между процессами и хранятся в локальной памяти взаимодейст-

вующих процессов. Передача сообщений между процессами может осуществляться синхронно или асинхронно.

В синхронной модели два процесса при взаимодействии сначала синхронизируются, затем обмениваются данными и после этого продолжают асинхронно выполняться. Такой механизм взаимодействия и синхронизации процессов называется *рандеву*. Существует рандеву *простое* и *расширенное*. При простом рандеву обеспечивается односторонний обмен информации от источника к получателю. Расширенное рандеву, или транзакция, обеспечивает двухстороннюю передачу информации с использованием одной синхронизации. После установления рандеву производится копирование информации от запрашивающего процесса-клиента к процессу-получателю или серверу. Затем процесс-клиент приостанавливается до выполнения процессом-сервером запроса. После этого результаты пересылаются к процессу-клиенту, который после этого может продолжать работу.

Современные ЯПП, ориентированные на реализацию в среде мультипроцессорных ВС, используют модель взаимодействия процессов, основанную на передаче сообщений. Известно три основных подхода к созданию языков параллельного программирования.

1. Введение в состав традиционных последовательных языков программирования (например, Фортран, Си и др.) средств поддержки параллелизма, осуществляемых на уровне языковых конструкций или в виде библиотечных функций. Преимуществом такого подхода является широкая распространенность языка, знакомство с ним пользователей, что облегчает освоение новшеств языка. Недостаток – противоречия концепций базового языка с новыми средствами, мешающие и затрудняющие их эффективную реализацию.

2. Создание оригинальных языков параллельной обработки, ориентированных на конкретную архитектуру или тип машин, например язык Occam для транспьютеров. Преимущество такого подхода состоит в возможности создании гибких и выразительных языковых (параллельных) конструкций, а также их эффективной реализации. Недостаток – в необходимости изучения этого языка, трудности переноса программы на другие типы машин. Такие языки редко получают широкое распространение.

3. Создание новых языков программирования, независимых от конкретной архитектуры или типа машины (например, Ada). Преимущество третьего подхода – в возможности введения новейших концепций программирования, повышающих эффективность ЯПП и надежность написанных на нем программ. Основной недостаток – необходимость изучения множеств новых концепций и языковых конструкций, сложность создания эффективных объектных кодов программ при трансляции.

Можно выделить типичные языковые средства для описания асинхронных параллельных процессов, выполняемых на МВС. К ним относятся: средства описания процессов, средства инициации и завершения процессов, взаимодействия и синхронизации процессов, описание топологии сети процессов и др.

В ЯПП единицей параллелизма является элементарный процесс, который по своей природе является последовательным и может выполняться параллельно с другими процессами. Процесс может быть описан как блок, функция, подпрограмма, задача.

В ЯПП, базирующихся на модели взаимодействия процессов через передачу сообщений, используются два типа языковых конструкций.

1. Примитивы синхронной передачи сообщений на базе простого рандеву (односторонний обмен информации) типа:

SEND < сообщение > TO < процесс - получатель >
WAIT < сообщение > FROM < процесс - источник >,

где < сообщение > – данные, передаваемые в сообщении; <процесс-получатель> – идентификатор процесса-получателя сообщения; < процесс-источник > – идентификатор процесса-отправителя сообщения.

Процесс-источник посылает сообщение, исполняя оператор SEND, процесс-получатель ждет прибытия посланного сообщения при выполнении оператора WAIT. Указанные примитивы объединяют синхронизацию с передачей сообщений. Синхронизация происходит, когда два процесса готовы к обмену сообщениями, т.е. один из взаимодействующих процессов, находящийся в состоянии исполнения оператора SEND/WAIT, совпадает с симметричным

состоянием процесса-партнера, исполняющего соответствующий оператор WAIT/SEND. Если эти состояния не совпадают, то процессы переводятся в состояние ожидания.

2. Высокоуровневая конструкция, называемая «вызовом удаленной процедуры», реализуется расширенным рандеву. Эта конструкция используется в языке Ada. Языковая конструкция вызова удаленной процедуры аналогична вызову традиционной процедуры. В данном практикуме этот механизм не изучается.

При выполнении параллельной программы в мультипроцессорной среде необходимы языковые средства описания физической конфигурации процессоров, логической структуры (топологии) процессов, отображения компонент параллельной программы на процессорах. В общем случае число процессов может превышать число процессоров и должна быть обеспечена возможность отображения нескольких процессов на один процессор. Распределение вычислительных ресурсов между параллельными процессами не влияет на логику программы. Существуют различные технологии использования языковых средств конфигурирования. В одном случае описание конфигураций совмещается с определением программы и является частью исходного текста программы, в другом случае определение конфигурации отделено от определения процессов и не входит в исходный текст программы. В первом случае средства конфигурации входят в состав ЯПП, во втором случае они оформлены в самостоятельный язык описания конфигурации. Первый подход реализован в языке Occam, второй – в языке 3L Параллельный Си.

2.2. Язык Параллельный Си

Для разработки параллельных программ в настоящее время существуют различные языки и системы программирования: Occam, Си, Паскаль, Фортран, Ada и др. Из перечисленных языков наибольшее распространение получили языки Си и Фортран.

Имеется несколько версий систем программирования языка Си. Все они обеспечивают генерацию объектного кода. Их основное отличие в различных возможностях поддержки взаимодействия параллельных процессов.

Система программирования Параллельный Си фирмы 3L (Lattice Logic Limited) [7], с которой предстоит познакомиться в рамкам данного практикума, при программировании параллельных процессов не требует обязательного использования языка Occam, однако не исключается возможность их комбинированного использования. Дальнейший материал будет содержать описание средств поддержки параллелизма языка Параллельный Си-3L.

Операционная среда системы программирования Параллельный Си-3L – комплекс программно-технических средств, обеспечивающих поддержку процесса разработки прикладных программ и их эксплуатацию. По составу технических средств она является разнородной системой, включающей управляющий HOST-компьютер и сеть объектных процессоров, один из которых (будем его называть ROOT) связан с HOST-компьютером каналом-линком (компортом). Сеть процессоров служит средой исполнения инструментальных средств и прикладных программ. Межплатная связь каналов процессоров обеспечивается кабелями.

Разработка прикладных программ обеспечивается инструментальными средствами системы программирования Параллельный Си, функционирующими в среде управляющей ПЭВМ и на процессорах.

При эксплуатации прикладных программ управляющая ПЭВМ используется для загрузки компонент (задач) программы в процессорную сеть, а также в качестве источника разделяемых ресурсов (клавиатуры, экрана, функций MS-DOS, файлов) для компонент прикладной программы.

Компилятор системы Параллельный Си-3L обеспечивает полную реализацию языка Си стандарта ANSI. На уровне последовательных языковых конструкций основными отличиями языка Параллельный Си-3L (далее обозначаемого как 3L-C) ANSI-C являются:

- увеличенная длина идентификатора до 31 символа;
- разрешение использовать имени структур данных вместо указателя на структуру в операциях присваивания и аргументах функций;
- введение нового типа VOID, используемого для описания функций, не имеющих возвращаемых значений;

- введение оператора вставки ассемблерных команд в исходный текст программы.

Различия на уровне библиотечных функций предопределены стандартом ANSI-C, по которому состав библиотечных функций не фиксирован. В 3L-C имеются две библиотеки стандартных функций: обычная (run-time) и автономная (standalone). Обычная библиотека используется в программах, работающих на корневом процессоре и обменивающихся данными с управляющей PC (HOST). Автономная библиотека используется в программах, работающих на процессорах, соединяемых только с соседними процессорами. Автономная библиотека является подмножеством функций обычной библиотеки. В автономной библиотеке устранены те функции, которые осуществляют операции ввода/вывода через HOST-компьютер. Кроме того, в кодах main-функций задач, работающих в узлах процессорной сети, не нужны операции ввода и обработки параметров командной строки, являющихся обязательной компонентой кодов функций задач, работающих на корневом процессоре. В результате объем кодов отдельных задач, как и всей прикладной программы, удается значительно сократить.

Обычная библиотека состоит из следующих групп стандартных функций:

- потокового ввода/вывода для файлов и устройств, включая средства прямого доступа к файлам;
- низкоуровневого ввода-вывода блоков данных для файлов и устройств;
- математических функций;
- манипуляций строками;
- определения принадлежности символа определенному классу;
- преобразования числовых значений из машинного представления в символьное и наоборот;
- динамического распределения памяти;
- определения даты и времени;
- доступа к функциям MS-DOS;
- пакета THREAD;
- пакета работы с семафорами;

- доступа к таймеру процессора;
- обмена через каналы процессора;
- маршрутизации передачи сообщений по сети процессоров;
- синхронизации доступа к библиотечным функциям при работе с THREAD;
- прочих функций.

В состав библиотек стандартных функций входят файлы стандартных спецификаций библиотечных функций, которые необходимо использовать в программе путем включения соответствующего файла спецификации через оператор препроцессора #include. В языке 3L-C расширен состав параметров main()-функции.

Главная функция вызывается со следующими параметрами:

```
typedef int CHAN;
main(argc,argv,envp,in,inlen,out,outlen)
int argc,inlen,outlen;
char * argv [], * envp [];
CHAN * in [], * out [];
```

где argc содержит количество параметров командной строки; argv – массив значений параметров командной строки, включая имя main()-функции; envp всегда равен NULL; in, out – указатели векторов процессорных каналов ввода/вывода; inlen, outlen – число элементов в in и out соответственно.

Полный формат main()-функции используется при вызове программ, работающих с функциями обычной библиотеки. Для программ, работающих с функциями автономной библиотеки, argc = 1, argv[1] = NULL, т.е. передача аргументов через командную строку в этом случае не осуществляется.

В системе 3L-C языковые средства описания топологии параллельной программы, конфигурации сети процессоров и отображения компонент программы на процессоры выделены в автономный язык описания конфигурации. Описание этого языка приведено в разд. 2.3. Конфигурация программы описывается в виде файла и обрабатывается утилитами CONFIG или FCONFIG, генерирующими загрузочный файл прикладной программы на сеть процессоров.

Концептуальной основой языка 3L-C является абстрактная модель языка Occam. В соответствии с этой моделью вычислительная система описывается набором конкурентных процессов, взаимодействующих между собой через процесс-посредник, называемый *каналом*. Канал связывает только два процесса – источник и получатель, и не имеет очередей. Канал является односторонним, и для взаимодействия процессов в двух направлениях необходимо использование двух каналов. Канал совмещает функции передачи сообщения с функцией синхронизации взаимодействующих процессов на основе механизма простого рандеву. Процесс в этой модели представляется «черным ящиком», связанным с внешним миром через свои входы и выходы, выполняющим некоторую последовательность действий.

Программа в 3L-C описывается как набор конкурентно-выполняемых задач. Каждая задача имеет вектор входных и выходных портов, посредством которых обеспечивается их взаимодействие. На абстрактном уровне задача рассматривается как «черный ящик», входам и выходам которого соответствуют порты.

Задачи являются элементами (модулями) для построения параллельных программ. Наиболее типовые, стандартные задачи, оформляются в виде библиотек и поставляются в составе компиляторов.

Каждая задача описывается как автономная программа, содержащая main()-функцию, при вызове которой в качестве аргументов могут передаваться векторы входных и выходных портов. Каждый элемент векторов портов описывается как указатель к слову канала (CHAN*).

Каждой задаче выделяется область памяти для кодов и данных. Задача выполняется на одном процессоре, программа – на одном или на нескольких процессорах. Программа как набор взаимосвязанных задач описывается средствами языка конфигурации. При этом специфицируются имена задач и топология программы, т.е. схема соединения задач посредством портов. Язык конфигурации специфицирует также привязку задач к физическим процессорам, на которых они будут выполняться, а также привязку портов задач к физическим каналам процессоров, если эти задачи выполняются на различных процессорах. Каждый процессор обеспечивает выполнение на нем любого числа задач; ограничения определяются объемом оперативной памяти.

Привязка портов задачи к каналам процессоров реализуется библиотечными функциями передачи сообщений по каналам. Задачи, работающие на одном процессоре, могут взаимодействовать через неограниченное число каналов. В этом случае каналы реализуются через внутреннюю память процессора.

В составе 3L-C нет языковых конструкций инициации задач-компонент параллельной программы. Инициация задач производится инициацией программы в целом и осуществляется специальной программой server.

Завершение всей программы в целом обеспечивается функцией exit(). Эта функция может использоваться только в теле корневого процесса и неприемлема для узловых процессов.

Для обмена сообщениями между процессами, выполняющимися на процессоре, используются библиотечные функции пакета CHAN, реализующие передачу сообщений (байт, слово, пакет) через каналы процессоров. Синхронизация задач обеспечивается при использовании библиотечных функции CHAN (через механизм каналов) и/или пакета SEMA (через механизм семафоров).

В целом язык и система программирования 3L Параллельный Си предоставляют все необходимые средства и возможности как для экспериментального изучения проблематики многопроцессорных ВС и параллелизма, так и для разработки прикладных параллельных процессорных программ, имеющих практическую значимость.

2.3. Язык конфигурации

Язык конфигурации 3L-C обеспечивает наиболее простой способ описания конфигурации физической сети процессоров и пользовательских задач, не требующий от пользователей знания тонкостей загрузки задач на сеть процессоров. Входной файл программы на языке конфигурации состоит из последовательности операторов, начинающихся с новой строки.

Множество операторов, используемых во входном файле:

statement =
processor statement
wire statement

task statement
connect statement
place statement
bind statement

Замечание. Операторы могут появляться во входном файле в любом порядке, но использовать какой-либо объект конфигурационного языка (processor, wire, task) можно только после того, как он будет объявлен!

Ниже приведены описания основных операторов языка описания конфигурации.

Оператор PROCESSOR

processor statement =
"PROCESSOR", new identifier, { processor attribute };

processor attrlbute =
"TYPE", "=", processor type;

processor type =
"PC";

Оператор PROCESSOR декларирует использование физического процессора. Все процессоры сети, используемые в задаче, должны быть декларированы этим оператором, включая HOST-процессор, через который осуществляется загрузка сети (обычно это IBM PC). Мнемоническое название host конфигуратор воспринимает как загрузочный процессор. Таким образом, конфигурационный файл должен включать в себя следующий оператор:

processor host

Процессоры декларируются в конфигурационном файле прежде, чем на них будет размещена пользовательская задача.

Атрибутом TYPE можно сообщить конфигуратору, что данный процессор уже загружен. Например, физическая сеть, содержащая один процессор и два IBM PC, описывается следующим образом:

```
processor host
processor root_processor
processor other_IBM_PC type=PC
```

По умолчанию host-ЭВМ всегда имеет type=PC, все остальные подразумеваются как процессоры сети.

Предполагается, что каждый процессор в сети имеет линки. На их номера можно ссылаться (оператором WIRE) посредством спецификатора линка, который состоит из идентификатора процессора и номера линка, заключенного в квадратные скобки:

```
link specifier =
processor identifier, "[",constant,"]";
```

Например, линк номер 3 процессора, имеющего мнемоническое имя extra, может быть специфицирован как extra[3].

Оператор WIRE

Формат оператора:

```
wire statement =
"WIRE", new identifier, link specifier, link specifier;
```

Оператор WIRE служит для спецификации физического провода, связывающего процессорные линки. Каждый провод поддерживает две связи – по одной в каждом направлении. Поэтому без изменения смыслового содержания любой провод может быть описан двумя способами, например, один и тот же провод с мнемоническим названием yellow_wire, связывающий линк 2 процессора proc_one с линком 3 процессора proc_two, можно описать двумя способами;

```
wire yellow_wire proc_one[2] proc_two[3]
wire yellow_wire proc_two[3] proc_one[2]
```

Оператор TASK

Формат оператора:

```
task statement =
      "TASK", new identifier, {task attribute };
task attribute =
"INS", "=",constant I
"OUT","=", constant I
"FILE", "=", task file specifier
"OPT", "=", opt area I
"URGENT" I
memory area, "=",memory amount;
opt area =
      memory area | "CODE";
memory area =
"STACK" I "HEAP" I "STATIC" I "DATA" ;
memory amount =
constant I "?";
task file specifier =
identifier f string constant;
```

Оператор TASK служит для объявления задачи, которая может быть либо пользовательской, либо одной из стандартных задач. Оператор может содержать несколько атрибутов, каждый из которых описывает некоторую особенность задачи. Очередность появления атрибутов в операторе произвольна.

Атрибут INS. Обязательный атрибут, который специфицирует число элементов в векторе входного порта задачи. Даже если задача не имеет входных портов, необходимо специфицировать число входных портов как ins=0.

Атрибут OUTS. Обязательный атрибут, который специфицирует число элементов в векторе выходного порта задачи. Даже если задача не имеет выходных портов, необходимо специфицировать число выходных портов как outs=0.

Атрибут FILE. Необязательный атрибут, специфицирующий файл, в котором находится объектный код задачи, создаваемый редактором связей.

Атрибут размера памяти. Этот атрибут служит для спецификации размера памяти, используемой в качестве рабочего пространства для задачи, а также определения стратегии выделения этой памяти.

Аргументом одного из атрибутов data является целочисленное выражение числа затребованных байт памяти. Размер памяти менее 128 байт не воспринимается. Это сделано для того, чтобы предотвратить случайный ошибочный ввод недопустимо малых значений, например случайно записать 1.6 вместо 1.6К. Можно специфицировать значение всей оставшейся памяти в процессоре введением вместо целого числа знака "?". Однако только для одной задачи на процессоре можно задать требование к памяти в такой форме.

Предусмотрено два типа стратегии распределения памяти.

Стратегия типа single-vector используется в случае задания атрибута data. При этой стратегии задача использует одну общую область памяти для всего затребованного рабочего пространства, где размещает static, stack, heap данные. Stack и heap данные располагаются в памяти в противоположных концах, нарастая к противоположной стороне.

Пример:

```
task x . . . data=50к. . .
```

Стратегия double-vector используется, если заданы атрибуты stack и heap. При этой стратегии стек занимает отдельную область памяти от всего рабочего пространства задачи, что позволяет, в случае если он небольшой, размещать стековые переменные во внутрикристалльной памяти процессора. Это, в свою очередь, позволяет существенно повышать скорость вычислений.

Пример:

```
task x . . . stack=1k heap=10k
```

Стратегии взаимно исключают друг друга. Другими словами, если специфицирован атрибут DATA, то тогда уже не нужно специфицировать атрибуты stack и heap.

Если атрибуты размера памяти вообще не заданы, то по умолчанию будет воспринято как DATA = ?. Другими словами, осуществляется стратегия single-vector в оставшейся памяти процессора.

Атрибут OPT. Этот атрибут специфицирует то, что область памяти, заданная в качестве его аргумента, должна быть выделена, по возможности во внутрикристалльной памяти процессора.

Атрибут URGENT. Необязательный атрибут, он специфицирует то, что начальный thread задачи будет запускаться с приоритетом уровня URGENT (срочный). По умолчанию: not urgent.

Оператор CONNECT

Оператор CONNECT объявляет использование логического канала между портами двух задач.

```
connect statement =
"CONNECT", new identifier, output port specifier,
input port specifier;
output port specifier =
port specifier;
input port specifier =
port specifier;
```

Например, связь между 0 портом задачи server и 0 портом задачи filter описывается следующим образом:

```
connect ? serverl[0] filter[0];
connect ? filter[0] server[0]
```

Замечание. Тип порта (входной или выходной) определяется порядком (контекстом) расположения портов в операторе CONNECT, этот порядок в отличие от оператора WIRE принципиален.

Оператор PLACE

Оператор PLACE устанавливает процессор, на котором будет выполняться задача. Формат оператора:

"PLACE", task identifier, processor identifier;
processor identifier =
identifier;
task Identifier =
identifier;

Пример использования оператора:

```
place user_task root
place server host
```

2.4. Пример создания приложения и конфигурационного файла

Пусть приложение состоит из четырех задач h1, h2, h3, h4. Задача h1 располагается на Root-процессоре, h2 – на втором, h3 – на третьем, h4 – на четвертом процессорах. Процессоры объединены в кольцо. Задачи связаны одна с другой одним каналом.

Последовательность команд
для создания и выполнения приложения

```
command /c c40c h1 h2 h3 h4

command/c c40ctask h1
command/c c40cstask h2
command/c c40cstask h3
command/c c40cstask h4

config h.cfg h.app

tis h.app
```

Содержимое файла h.cfg

```
!Hardware configuration
! total of 4 processors found
processor ROOT type=TMS320C40
processor P1 type=TMS320C40
```

```
   wire ? P1[5] ROOT[2]
processor P2 type=TMS320C40
   wire ? P2[5] P1[2]
processor P3 type=TMS320C40
   wire ? P3[2] ROOT[5]
   wire ? P3[5] P2[2]

! Software configuration

!task h1
!task h2
task h1 ins=2 outs=2
task h2 ins=2 outs=2
task h3 ins=2 outs=2
task h4 ins=2 outs=2

place h1 ROOT
place h2 P1
place h3 P2
place h4 P3

connect ? h1[0] h2[0]
connect ? h2[0] h1[0]

connect ? h2[1] h3[0]
connect ? h3[0] h2[1]

connect ? h3[1] h4[0]
connect ? h4[0] h3[1]

connect ? h4[1] h1[1]
connect ? h1[1] h4[1]
```

2.5. Технология работы в дисплейном классе

Выполнение практикума производится на многопроцессорной вычислительной системе. На материнской плате этой системы установлено несколько вычислительных модулей с сигнальным процес-

сором TMS320C40, коммуникационные порты которых соединены в кольцо, как показано на рис. 2.1. Второй компорт каждого модуля соединен с пятым компортом другого модуля. Первый вычислительный модуль (ROOT) является главным и управляет работой всех остальных, его третий компорт соединен с PCI интерфейсом PC.

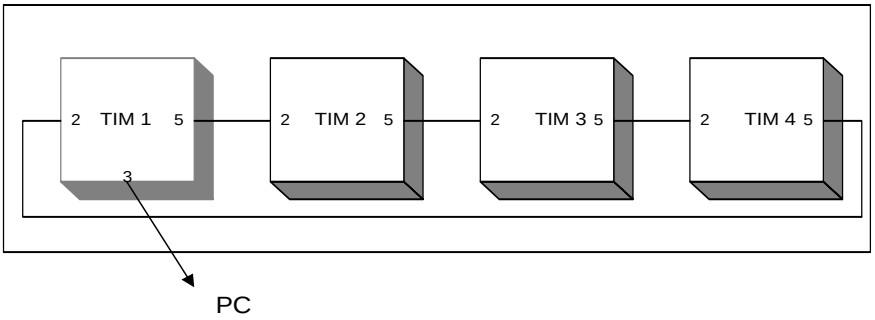

Рис. 2.1. Конфигурация системы для лабораторной работы 1

Для удаленного коллективного доступа к многопроцессорным вычислительным ресурсам из дисплейного класса используется клиент-серверное приложение. С его помощью возможно выполнение параллельных программ в пакетном режиме. Множество задач, предназначенных для параллельного выполнения, в совокупности с конфигурационным файлом и исходными данными составляют задание. Система поддержки обеспечивает переадресацию стандартного ввода, вывода и протокола задания в файлы. Результат выполнения задания пересылается в файл, в который был переадресован стандартный вывод и протокол.

Серверная часть системы реализована в виде приложения tms.exe, которое является универсальной оболочкой для запуска однопользовательских утилит (компилятора, редактора связей, конфигуратора задач, поставляемых разработчиком системы). Предполагается, что обработка задания состоит из двух стадий. На первой стадии запускается командный файл. Если его выполнение прошло без ошибок, то на второй стадии запускается исполняемый файл, обеспечивающий доступ к многопроцессорной вычислительной системе. Серверное приложение разработано на языке Си с использованием только

API-функций в среде Visual C++ 5.0. Приложение может работать в операционной системе Windows 95/98/NT.

Клиентское приложение функционирует в среде Linux и представляет собой набор утилит командной строки и shell-процедур. Для выполнения задания необходимо создать каталог, в который помещаются все файлы этого задания. В указанном каталоге не должно быть файлов, не имеющих отношения к заданию. При одновременной работе с несколькими заданиями необходимо создать для каждого задания свой каталог (каталог задания). Конфигурационный файл должен называться h.cfg. Ввод с клавиатуры при помощи библиотечной функции scanf автоматически переадресуется из файла с именем infile, в который предварительно необходимо поместить входные данные. Ввод и вывод данных процедурами scanf, printf может производиться только из корневой задачи. Имена остальных файлов произвольны. Соответствие между именами исходных файлов на языке Си и процессорами задается в конфигурационном файле. Рекомендуется текст корневой задачи поместить в файл с именем h1.c, и остальным задачам присваивать имена, соответственно, h2.c, h3.c, h4.c.

Для программирования задач разработчиком системы поставляется компилятор Параллельного Си, включающий в себя стандартный Си (ANSI) и дополнительные функции для параллельных вычислений, обеспечивающие межмодульную передачу данных. Для каждой задачи назначается свой вычислительный модуль и устанавливаются логические каналы обмена данными. Аппаратная конфигурация не накладывает никаких ограничений на топологию логических связей между задачами, так как программный конфигуратор обеспечивает разделение одного физического канала несколькими логическими каналами, а также позволяет создавать логические каналы между непосредственно не связанными вычислительными модулями. Синхронизация задач происходит во время передачи данных по логическим каналам. Чтение данных из канала возможно в момент записи данных другой задачей в этот канал. В противном случае одна из задач переводится в состояние ожидания до момента выполнения операций чтения/записи другой задачей.

Запуск задания на выполнение осуществляется командой:

all <имя каталога задания>

Запускать новое задание можно только после получения результатов предыдущего. Получить результаты выполнения задания можно при помощи команды result. Прием новых заданий на выполнение заблокирован до получения результатов предыдущего задания. При попытке запуска второго задания до получения результатов предыдущего выводится соответствующее сообщение.

Если выходные данные еще не готовы, необходимо повторно ввести команду result спустя несколько секунд. Отсутствие результатов выполнения некоторого задания более трех минут может означать возникновение системной ошибки. В этом случае необходимо обратиться к преподавателю и строго следовать его указаниям.

Результат выполнения задачи представляет собой переадресованные в файл стандартный вывод и протокол, т.е. результат выполнения функций printf и fprintf(stderr, …).

2.6. Методические указания по отладке и выполнению параллельной программы

Отладка параллельных программ в общем случае производится в два этапа.

На первом этапе осуществляется автономная отладка компонент (задач) параллельной программы с использованием большинства приемов отладки последовательных программ.

На втором этапе выполняется комплексная отладка всей программы. Наиболее простой прием отладки на этом этапе состоит во вставлении в текст программы дополнительных операторов, обеспечивающих вывод отладочных сообщений на экран или принтер HOST-ЭВМ.

Рассмотрим подробнее порядок работы с клиентской частью системы коллективного доступа.

1. Необходимо создать каталог, в который в дальнейшем будут помещаться файлы задания. Все содержимое этого каталога подразумевается относящимся к одному заданию. В этом каталоге не

должно быть файлов, не имеющих отношения к заданию. При одновременной работе с несколькими заданиями необходимо создать несколько каталогов. В дальнейшем описании этот каталог называется *каталогом задания*. Конфигурационный файл должен называться h.cfg. Ввод с клавиатуры при помощи библиотечной функции scanf автоматически переадресуется из файла с именем infile, в который необходимо поместить входные данные задания. Ввод и вывод данных (scanf, printf) может производиться только из корневой задачи. Имена остальных файлов произвольны. Соответствие между именами исходных файлов на языке Си и процессорами задается в конфигурационном файле. Рекомендуем текст корневой задачи поместить в файл с именем h1.c, и остальным задачам присваивать имена, соответственно, h2.c, h3.c, h4.c.

2. После подготовки каталога задания (копирования в него всех необходимых файлов) необходимо осуществить запуск задания на выполнение командой all <имя каталога задания>. Запускать новое задание можно только после получения результатов предыдущего задания. В случае попытки запуска второго задания выводится соответствующее сообщение.

3. Получить результаты выполнения задания можно при помощи команды result. Выходные данные задания могут быть еще не готовы. В этом случае необходимо по истечении нескольких секунд повторно ввести команду result. Прием новых заданий на выполнение будет заблокирован до получения результатов предыдущего задания. Отсутствие результатов выполнения задания более трех минут означает возникновение системной ошибки. В этом случае необходимо обратиться к преподавателю и строго следовать его указаниям. Результат выполнения задачи представляет собой переадресованные в файл стандартный вывод и протокол, т.е. результат выполнения функций printf и fprintf(stderr, …).

4. Если в результате выполнения компиляции обнаружены синтаксические ошибки, то дальнейшая обработка задания прерывается. В этом случае в качестве результата передаются сообщения компилятора. Аналогичный результат получается в случае возникновения ошибок на этапе редактирования связей, например, если не найден один из файлов задачи, перечисленных в конфигурационном файле.

5. Система удаленного доступа снабжена средствами защиты от зацикливаний и бесконечных ожиданий: каждому заданию отводится строго определенное время выполнения на многопроцессорной системе. В случае превышения этого времени задание снимается с выполнения, при этом в результирующий файл помещается соответствующее сообщение. Такая ситуация чаще всего возникает при отсутствии синхронизации между задачами. Например, одна задача пытается прочитать данные из канала, в который другая задача не осуществляла записи. Если вторая задача так и не произведет запись в канал, то первая задача будет находиться в состоянии бесконечного ожидания. Заметим, что вывод библиотечной функции printf буферизуется, поэтому участок программы, в котором возникла такая ситуация, не всегда можно однозначно выявить при помощи отладочной печати. Отсутствие какого-либо сообщения отладки не означает, что бесконечное ожидание возникло до выполнения оператора вывода данного сообщения. Более точно можно локализовать место возникновения такой ситуации, если отладочную печать производить при помощи библиотечной функции fprintf(stderr, …), так как протокол обычно не буферизуется.

3. ОПИСАНИЯ ЛАБОРАТОРНЫХ РАБОТ*

3.1. Лабораторная работа 1. Знакомство с многопроцессорной системой TMS320C40 (вводная работа)

Цель: *изучение технологии работы с системой TMS320C40 в клиент/серверном режиме, изучение языка Параллельный Си, языка конфигурации и коммуникационных средств системы TMS320C40.*

3.1.1. Порядок выполнения работы

1. Составить общий последовательный алгоритм решения поставленной задачи.
2. Разделить общий алгоритм на части по числу доступных процессоров в системе. При этом надо стремиться к равномерному распределению объемов вычислений между отдельными процессорами. Следует также минимизировать объемы пересылаемой между процессорами информации. Все исходные данные должны вводиться в процессор ROOT через файл infile, все результаты также должны выводиться на экран процессором ROOT.
3. Подобрать наиболее подходящий вариант соединений процессоров для решаемой задачи.
4. Написать на языке Си отдельные программные модули для каждого процессора. Особое внимание следует обратить на согласование объемов передаваемой и принимаемой информации по каждому из программных каналов межмодульной связи.
5. Составить конфигурационный файл.
6. Отладить параллельную программу в соответствии с инструкцией и продемонстрировать ее работу преподавателю.
7. Ответить на контрольные вопросы

* Основная литература для подготовки и выполнения лабораторных работ в списке выделена курсивом.

3.1.2. Пример задания

Рассмотрим пример задания, состоящего из четырех задач, в котором корневая задача получает от остальных задач по целому числу и выводит их на экран. Пример файла конфигурации:

```
!Hardware configuration
! Эта часть задает тип процессоров и аппаратные связи между ними и изменению не подлежит.
! total of 4 processors found
processor ROOT type=TMS320C40
processor P1 type=TMS320C40
  wire ? P1[5] ROOT[2]
processor P2 type=TMS320C40
  wire ? P2[5] P1[2]
processor P3 type=TMS320C40
  wire ? P3[2] ROOT[5]
  wire ? P3[5] P2[2]

! Software configuration
! Программная конфигурация. Определяется студентом.
! Эта часть задает количество логических каналов каждой задачи.
task h1 ins=3 outs=0 data=50k
task h2 ins=0 outs=1 data=50k
task h3 ins=0 outs=1 data=50k
task h4 ins=0 outs=1 data=50k
! Эта часть задает распределение задач по процессорам.
place h1 ROOT
place h2 P1
place h3 P2
place h4 P3
! Эта часть задает соответствие задач и номеров логических каналов.
connect ? h2[0] h1[0]
connect ? h3[0] h1[1]
connect ? h4[0] h1[2]
```

Каждая задача в заданной конфигурации имеет связь с корневой задачей. Следует отметить, что логические каналы однонаправленные. Для задания двунаправленного канала следует определить две симметричные связи. Далее приведены тексты задач:

h1.c

```
#include <stdio.h>
#include <chan.h>

main(argc,argv,envp,in_ports,ins,out_ports,outs)
int argc,ins,outs;
char *argv[], *envp[];
CHAN *in_ports[], *out_ports[];
{
int a,i;
printf("Root begins\n");
for (i=0;i<3;i++){
      chan_in_word(&a,in_ports[i]);
      printf("a=%d\n",a);
      }
printf("Root ends\n");
}
```

h2.c

```
#include <stdio.h>
#include <chan.h>

main(argc,argv,envp,in_ports,ins,out_ports,outs)
int argc,ins,outs;
char *argv[], *envp[];
CHAN *in_ports[], *out_ports[];
{
int a;
a=2;
chan_out_word(a,out_ports[0]);
}
```

h3.c

```
#include <stdio.h>
```

```
#include <chan.h>

main(argc,argv,envp,in_ports,ins,out_ports,outs)
int argc,ins,outs;
char *argv[], *envp[];
CHAN *in_ports[], *out_ports[];
{
int a;
a=3;
chan_out_word(a,out_ports[0]);
}
```

h4.c

```
#include <stdio.h>
#include <chan.h>

main(argc,argv,envp,in_ports,ins,out_ports,outs)
int argc,ins,outs;
char *argv[], *envp[];
CHAN *in_ports[], *out_ports[];
{
int a;
a=4;
chan_out_word(a,out_ports[0]);
}
```

Как видно из приведенных текстов, обязательно подключение заголовочного файла chan.h. В нем содержатся описания специальных типов и прототипов функций работы с логическими каналами. Тип CHAN определяет логические каналы связи. Нумерация каналов в массиве соответствует нумерации в описании связей конфигурационного файла. Для передачи данных через логический канал используются две функции: chan_in_word(int *, CHAN *) – чтение данных из канала; chan_out_word(int, CHAN *) – запись данных в канал. Следует обратить внимание, что в операции чтения в качестве первого параметра передается адрес целой переменной, в которую производится чтение, а в операции записи – целая переменная. В случае передачи через канал вещественных данных необходимо выполнять явное преобразование типов. Например, необхо-

димо получить значение вещественного типа из in_ports[0] и передать его в out_ports[0] (f – переменная вещественного типа):

```
…
chan_in_word((int *)f, in_ports[0]);
chan_out_word(*(int *)&f, out_ports[0]);
…
```

3.1.3. Варианты заданий к лабораторной работе 1

Вариант 1

1. Даны матрицы A и B действительных чисел размерности $N \times N$.
2. Ввести N, A, B.
3. Вычислить $C = A \times B$.

Вариант 2

1. Даны векторы A и B действительных чисел размерности N.
2. Ввести N, A, B.
3. Вычислить скалярное произведение $C = A \times B$.

Вариант 3

1. Даны матрицы A и B действительных чисел размерности $N \times N$.
2. Ввести N, A, B.
3. Вычислить сумму $C = A + B$.

Вариант 4

1. Даны векторы B и $X0$ действительных чисел размерности N, матрица A действительных чисел размерности $N \times N$.
2. Ввести N, A, B, $X0$.
3. Решить уравнение $X = A \times X + B$ методом Гаусса.

Вариант 5

1. Даны векторы B и $X0$ действительных чисел размерности N, матрица A действительных чисел размерности $N \times N$.
2. Ввести N, A, B, $X0$.
3. Проверить сходимость итерационного процесса.
4. Решить уравнение $X = A \times X + B$ методом итераций с заданной точностью E.

Вариант 6

1. Даны A, B, E – действительные числа.
2. Подобрать функцию $F(X)$ из числа библиотечных, для которых на отрезке $[A, B]$ уравнение $F(X) = 0$ имеет один корень.
3. Ввести A, B, E.
4. Решить уравнение $F(X) = 0$ методом хорд с заданной точностью E.

Вариант 7

1. Даны A, B, E – действительные числа.
2. Подобрать функцию $F(X)$ из числа библиотечных, для которых на отрезке $[A, B]$ уравнение $F(X) = 0$ имеет один корень.
3. Ввести A, B, E.
4. Решить уравнение $F(X) = 0$ методом дихотомии с заданной точностью E.

Вариант 8

1. Дано: X – действительное число, N – целое число.
2. Ввести N, X.
3. Вычислить $S = X + X^2/2! - X^3/3! + \ldots + (-1)^N \times X^{(N+1)}/N!$.

Вариант 9

1. Даны A, B – действительные числа.
2. Подобрать функцию $F(X)$ из числа библиотечных, интегрируемую на отрезке $[A, B]$.
3. Ввести A, B, E.
4. Вычислить определенный интеграл функции $F(X)$ на отрезке $[A, B]$ с заданной точностью E методом трапеций.

Вариант 10

1. Даны A, B – действительные числа.
2. Подобрать функции $Y = F(X)$, $Y = \varphi(X)$ из числа библиотечных, для которых уравнение $Y = F(X) - Y = \varphi(X) = 0$ на отрезке $[A, B]$ имеет один корень.
3. Ввести A, B, E.
4. Найти координаты точек пересечения графиков заданных функций $Y = F(X)$, $Y = \varphi(X)$.

Вариант 11

1. Дано: X – действительное число, A – вектор действительных чисел размерности $N + 1$.
2. Ввести N, A.
3. Вычислить значение полинома

$$A[N] \times X^N + A[N-1] \times X^{(N-1)} + \ldots + A[0].$$

Вариант 12

1. Дан $Y = \exp(-X)$, где Y, X – векторы действительных чисел размерности N.
2. Ввести N, X, E.
3. Вычислить Y с заданной точностью E, разложив функцию Y в ряд Тейлора.

Вариант 13

1. Даны матрица A действительных чисел размерности $N \times N$, целое M.
2. Ввести N, A, M.
3. Вычислить $B = A^M$.

Вариант 14

1. Дана матрица A действительных чисел размерности $N \times N$.
2. Ввести N, A.
3. Вычислить определитель $||A||$ методом Гаусса.

Вариант 15

1. Дана матрица A действительных чисел размерности $N \times N$.
2. Ввести N, A.
3. Вычислить $C = A \times B$, где матрица B получена транспонированием матрицы A.

Вариант 16

1. Даны вектор B действительных чисел размерности N, матрица A действительных чисел размерности $N \times N$.
2. Ввести N, A, B.
3. Решить уравнение $A \times X = B$ методом Крамера.

Вариант 17

1. Даны векторы A, B и C действительных чисел размерности N.
2. Ввести N, A, B, C.
3. Найти $\sum_{i=1}^{N}(A_i \times B_i - C_i)$.

Вариант 18

1. Ввести действительные числа x и e.
2. Вычислить значение $\sin x$ при помощи библиотечной функции и разложением в ряд Тейлора.
3. Найти n – количество членов ряда, необходимое для вычисления функции с точностью e (e – относительная погрешность вычислений).

Вариант 19

1. Ввести действительные числа x и e.
2. Вычислить значение $\cos x$ при помощи библиотечной функции и разложением в ряд Тейлора.
3. Найти n – количество членов ряда, необходимое для вычисления функции с точностью e (e – относительная погрешность вычислений).

Вариант 20

1. Ввести действительные числа x и e.
2. Вычислить значение $\ln x$ при помощи библиотечной функции и разложением в ряд Тейлора.
3. Найти n – количество членов ряда, необходимое для вычисления функции с точностью e (e – относительная погрешность вычислений).

Вариант 21

1. Ввести действительные числа x и e.
2. Вычислить значение $\exp x$ при помощи библиотечной функции и разложением в ряд Тейлора.
3. Найти n – количество членов ряда, необходимое для вычисления функции с точностью e (e – относительная погрешность вычислений).

Вариант 22

1. Ввести действительные числа x и e.
2. Вычислить значение $\operatorname{tg} x$ при помощи библиотечной функции и разложением в ряд Тейлора.
3. Найти n – количество членов ряда, необходимое для вычисления функции с точностью e (e – относительная погрешность вычислений).

Вариант 23

1. Ввести действительные числа x и e.
2. Вычислить значение квадратного корня из x при помощи библиотечной функции и разложением в ряд Тейлора.
3. Найти n – количество членов ряда, необходимое для вычисления функции с точностью e (e – относительная погрешность вычислений).

Вариант 24

1. Даны матрицы A и B действительных чисел размерности $N \times N$.
2. Ввести N, A, B.
3. Вычислить $C = A \times A + B$.

Вариант 25

1. Дана матрица A действительных чисел размерности $N \times N$.
2. Ввести N, A.
3. Вычислить норму матрицы $\|A\|$.

Вариант 26

1. Дана матрица A действительных чисел размерности $N \times N$.
2. Ввести N, A.
3. Вычислить $\sum_i \sum_j A_{ij}$.

Вариант 27

1. Дана матрица A действительных чисел размерности $N \times N$.
2. Ввести N, A.
3. Вычислить $\min\limits_i \sum\limits_j A_{ij}$.

Вариант 28

1. Дана матрица A действительных чисел размерности $N \times N$.
2. Ввести N, A.
3. Вычислить $\min\limits_i \sum\limits_{j=1}^{n-1} (A_{ij+1} - A_{ij})$.

Вариант 29

1. Дана матрица A действительных чисел размерности $N \times N$.
2. Ввести N, A.
3. Вычислить $\prod\limits_{ij} A_{ij}^2$.

Вариант 30

1. Даны матрицы A, B действительных чисел размерности $N \times N$.
2. Ввести N, A, B.
3. Вычислить $\prod\limits_{ij} \left(A_{ij} - \min\limits_{ij} \left\{ B_{ij} \right\} \right)$.

Вариант 31

1. Даны матрицы A, B действительных чисел размерности $N \times N$.
2. Ввести N, A, B.
3. Вычислить $\min\limits_i \left\{ \prod\limits_j \left(A_{ij} - B_{ij} \right) \right\}$.

Вариант 32

1. Даны матрицы A, B действительных чисел размерности $N \times N$.
2. Ввести N, A, B.
3. Вычислить $\|C_{ij}\|$, где $C_{ij} = A_{ij} + B_{ij} - A_{ji}^2 - B_{ji}^2$.

Вариант 33

1. Даны матрицы A, B действительных чисел размерности $N \times N$.
2. Ввести N, A, B.
3. Вычислить $\|C_{ij}\|$, где $A_{ij} - \min\limits_i\{B_{ii}\} - \max\limits_i\{A_{ii}\} + B_{ij}$.

Вариант 34

1. Даны матрицы A, B действительных чисел размерности $N \times N$.
2. Ввести N, A, B.
3. Вычислить $\|C_{ij}\|$, где $C_{ij} = A_{ij} \times \left(B_{ij} - \max\limits_j\{A_{ij}\} \right)$.

Вариант 35

1. Даны матрицы A, B действительных чисел размерности $N \times N$.
2. Ввести N, A, B.
3. Вычислить скалярное произведение $F = D^*$, где $D_{ij} = \left(\min\limits_i\{A_{ij}\} \right)$, $E_{ij} = \left(\max\limits_j\{B_{ij}\} \right)$.

Вариант 36

1. Дана матрица A целых чисел размерности $N \times N$.
2. Ввести N, A.
3. Вычислить $C_{ij} = A_{ij}\ !\!- \min\limits_{ij}\{A_{ij}\}$.

Вариант 37

1. Даны матрицы A, B целых чисел размерности $N \times N$.
2. Ввести N, A, B.
3. Вычислить $C_{ij} = A_{ij}\,!/\,B_{ij}\,!$.

Вариант 38

1. Даны матрицы A и B действительных чисел размерности $N \times N$.
2. Ввести N, A, B.
3. Вычислить $C_{ij} = A_{ij}^{(N-i)} \,/\, B_{ij}^{(N-j)}$.

Вариант 39

1. Даны матрицы A и B целых чисел размерности $N \times N$.
2. Ввести N, A, B.
3. Вычислить $C_{ij} = A_{ij}^{B_{ij}} \,/\, B_{ij}^{A_{ij}}$.

Вариант 40

1. Даны матрицы A и B действительных чисел размерности $N \times N$.
2. Ввести N, A, B.
3. Вычислить $C_{ij} = A_{ij}\,!+\,B_{ji}\,!$.

Вариант 41

1. Даны матрицы A и B целых чисел размерности $N \times N$.
2. Ввести N, A, B.
3. Вычислить $C_{ij} = A_{ij}^{B_{ji}}$.

Вариант 42

1. Даны n векторов X_i размерности n каждый, X_i^j – целое.
2. Ввести N, X_i ($i = 1, \ldots, n$).
3. Проверить попарную ортогональность векторов X_i.

Вариант 43

1. Даны n ортогональных векторов X_i в n-мерном евклидовом пространстве.
2. Ввести n, X_i ($i = 1, \ldots, n$).
3. Проверить выполнение теоремы Пифагора для заданных векторов.

Вариант 44

1. Даны векторы X и Y в n-мерном евклидовом пространстве.
2. Ввести n, X и Y.
3. Найти косинус угла между векторами X и Y.

Вариант 45

1. Дана квазидиагональная матрица A действительных чисел размерности $N \times N$.
2. Ввести N, A.
3. Вычислить определитель матрицы A.

Вариант 46

1. Дана произвольная матрица A действительных чисел размерности $N \times N$.
2. Ввести N, A.
3. Вычислить определитель матрицы A методом обращения в 0 элементов строки.

Вариант 47

1. Дана произвольная матрица A действительных чисел размерности $N \times N$.
2. Ввести N, A.
3. Вычислить определитель матрицы A методом обращения в 0 элементов столбца.

Вариант 48

1. Дана произвольная матрица A действительных чисел размерности $N \times N$.
2. Ввести N, A.
3. Вычислить обратную матрицу A^{-1}.

Вариант 49

1. Дана произвольная матрица A действительных чисел размерности $N \times M$.
2. Ввести N, A, M.
3. Вычислить ранг матрицы A методом окаймления.

Вариант 50

1. Дана произвольная матрица A действительных чисел размерности $N \times N$.
2. Ввести N, A.
3. Вычислить определитель матрицы A методом представления в виде суммы определителей.

3.2. Лабораторная работа 2. Изучение технологии таймирования параллельной программы

Цель: *изучение технологии написания и отладки одного из алгоритмов расчетного типа с использованием векторной библиотеки для TMS320C40, практическое освоение технологии таймирования с целью оценки эффективности разработанной программы на различных топологиях MPP-сети процессоров TMS320C40.*

3.2.1. Способы временных оценок параллельных программ

В данном разделе рассмотрены два способа экспериментальной оценки эффективности параллельной программы и производительности многопроцессорных (параллельных) вычислительных систем (ВС).

Способ 1. Первый, наиболее часто используемый на практике способ, основан на вычислении отношения производительности параллельной ВС, которую она показывает при выполнении данной задачи, и производительности для данной же задачи при ее однопроцессорной реализации. Следуя [10], будем называть ускорением параллельной программы на данной ВС отношение, которое вычисляется в общем случае как:

$$R(p) = p(s)/p(1),$$

где R – коэффициент ускорения; $p(s)$ – производительность ВС, состоящей из s процессоров (вычислительных узлов) для данной параллельной программы, $s = 1, 2, \ldots, n$, где n – максимальное число процессоров (вычислительных узлов) в данной ВС; $p(1)$ – экспериментально измеренная производительность ВС, состоящей из одного процессора (вычислительного узла). Данное общее выражение ускорения используется для аналитической оценки ускорения различных архитектур [10].

Следует подчеркнуть, что данный критерий – функция, где аргументом является число процессоров/вычислительных узлов, что позволяет строить график этой зависимости.

На практике не очень удобно измерять или оценивать производительность ВС, поэтому более употребительным является измере-

ние времени выполнения программы (таймирование) и вычисление ускорения (speedup), см. например [11]:

$$S(p) = t(1)/t(p),$$

где S – тот же коэффициент ускорения; $t(p)$ – экспериментально измеренное время выполнения параллельной программы на ВС, состоящей из p процессоров (вычислительных узлов), $p = 1, 2, \ldots, P$, где P – максимальное число процессоров (вычислительных узлов) в данной ВС; $t(1)$ – экспериментально измеренное время выполнения программы на ВС, состоящей из одного процессора (вычислительного узла).

Способ 2. Наряду с построением графика зависимости коэффициента ускорения для параллельной программы на данной ВС от числа процессоров, можно оценивать «степень параллельности» конкретной архитектуры при решении векторных задач.

В книге [5] предложен унифицированный критерий для оценки производительности векторных операций для различных архитектур.

Общая формула для данного критерия выглядит так:

$$t = r_\infty^{-1} \times (N + n_{1/2}),$$

где t – время, приходящееся на выполнение операции (функции/алгоритма) с вектором длиной N; r_∞ – максимальная или асимптотическая производительность (для обычных компьютеров она достигается на векторах бесконечной длины), обычно быстродействие измеряют в Mflops (миллионах операций с плавающей запятой в секунду); $n_{1/2}$ – длина полупроизводительности, т.е. длина вектора, на которой достигается половина максимальной производительности.

Следуя [5], отметим, что смысл этих двух параметров совершенно различен: r_∞ характеризует технологические аспекты конкретной ВС, например повышение тактовой частоты процессора приведет только к изменению наклона экспериментальной прямой, но никак не отразится на величине полупроизводительности.

Длина же полупроизводительности представляет собой параметр для оценки степени параллелизма в архитектуре конкретной ВС для данной задачи, причем он меняется от 0, для последова-

тельной архитектуры (без распараллеливания), до бесконечности, для бесконечной процессорной матрицы. Следовательно, этот критерий дает количественную однопараметрическую оценку степени параллелизма конкретной архитектуры при решении конкретной векторной задачи и позволяет сравнить насколько «параллельным» является ВС для этой задачи. Чем выше значение длины полупроизводительности, тем более эффективной следует считать конкретную ВС для данной задачи.

На рис. 3.1 показана графическая интерпретация данной зависимости для случая идеального векторного вычислителя.

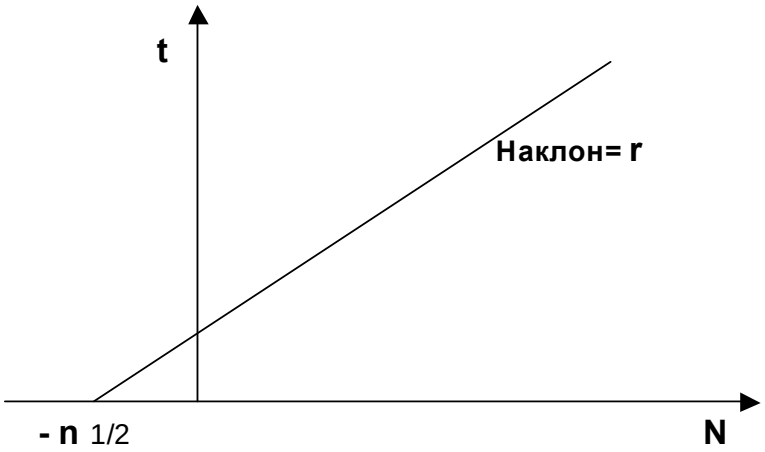

Рис. 3.1. Иллюстрация унифицированного критерия производительности векторных операций

Построив на основе экспериментальных данных, график такой зависимости можно без труда оценить как коэффициент $n_{1/2}$, т.е. длину полупроизводительности, которая на графике равна величине «пятки» данной прямой при $t = 0$, так и величину r_∞ – максимальную или асимптотическую производительность, которая равна обратному значению тангенса угла наклона этой эмпирической прямой. Однако если график построенной экспериментальной кривой значительно отличается от прямой, то это означает, что для данной вычислительной системы такой критерий не является адекватным и целесообразность/корректность его использования требует осмысления!

Данный критерий целесообразно использовать при решении вычислительных алгоритмов на различных аппаратных архитектурах, в том числе и на MPP-системах, в тех случаях, когда возможно экспериментальное исследование производительности параллельной программы или типовой подоперации параллельной программы в зависимости от длины исходных векторных данных. Построив эмпирический график такой зависимости, можно, во-первых, качественно оценить «векторность» данной архитектуры для данного алгоритма (по наличию и величине «отрицательной пятки» этого графика, а также его линейности) и, во-вторых, получить грубые оценки пиковой производительности для данной задачи на конкретной ВС. Практическое применение данного критерия целесообразно проводить для сравнения вариантов реализации программ, содержащих векторные вычисления, и сравнения полученных экспериментальных значений $n_{1/2}$.

3.2.2. Рекомендуемые топологии

В данной лабораторной работе следует ориентироваться на одну из четырех различных мультипроцессорных конфигураций (рис. 3.2 – 3.5).

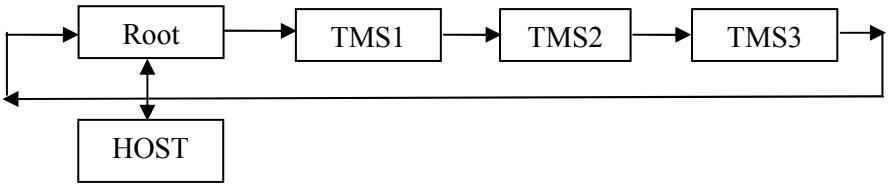

Рис. 3.2. Топология 1. «Конвейер – кольцо» из четырех процессоров

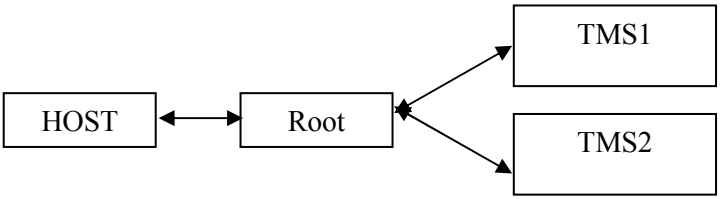

Рис. 3.3. Топология 2. Древовидная топология из трех процессоров

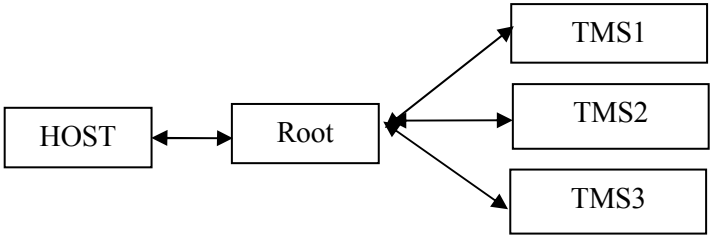

Рис. 3.4. Топология 3. Древовидная топология из четырех процессоров

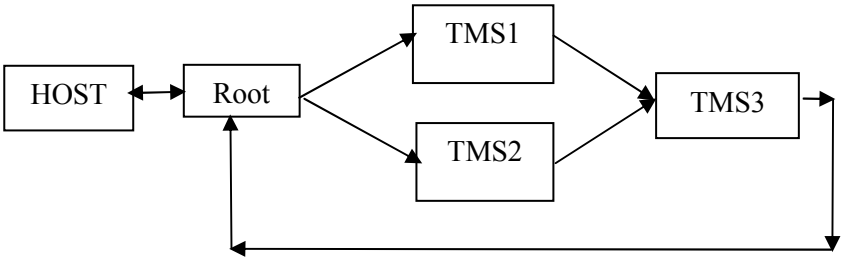

Рис. 3.5. Топология 4. Матрично-конвейерная топология из четырех процессоров

3.2.3. О векторной библиотеке Veclib

Для эффективной поддержки векторных вычислений фирмой Sinectonalysis Inc. разработана достаточно обширная библиотека прикладных программ-функций Veclib, которая оптимизирована под процессор TMS320C40 и предназначена для поддержки и выполнения векторных операций и алгоритмов. Основными группами алгоритмов, поддерживаемых данной библиотекой, являются:

- алгебраические функции (корень квадратный, экспонента, логарифм, тригонометрические функции);
- логические операции над векторами;
- операции над векторами (нормы, средние, поиск максимумов, сравнение, копирование, индексирование, сравнение векторов и т.п.);
- арифметические векторные операции, причем с результатом как типа вектор, так и типа скаляр;

- DSP (Digital Signal Processing) вычисления (фурье- и спектральный анализы, фильтрация);
- обработка изображений (фурье-преобразование, корреляции);
- алгоритмы сжатия;
- интерполяция, интегрирование и некоторые др.

Для использования функций из этой библиотеки необходимо объявить в своей программе заголовочный файл:

<dspvec.h>

В прил. 2 приведены некоторые функции из библиотеки Veclib с кратким описанием их функционала, а также прототипа, списка и типов аргументов.

3.2.4. Методические указания к лабораторной работе 2

1. Для обмена сообщениями между процессами, выполняющимися на MPP-сети процессоров, используются библиотечные функции пакета CHAN, реализующие передачу сообщений (байт, слово, пакет) через каналы процессоров.

Для построения приложения рекомендуются следующие библиотечные функции, для использования которых необходимо подключение заголовочного файла <chan.h> [3]:

chan_in_word() – ввод слова из входного порта;

chan_out_word() – вывод слова в выходной порт;

chan_in_message() – ввод сообщения из входного порта;

chan_out_message() – вывод сообщения в выходной порт.

Здесь имеются в виду порты задач. Между портами двух задач имеется логический канал. Например, задача h1 соединяется логическим каналом с задачей h2 через свой 0-й выходной порт (out_ports[0]) и входной порт 0 задачи h2 (in_ports[0]).

2. При применении способа 1 измерения эффективности параллельной программы на практике в каждом конкретном случае необходимо анализировать структуру ВС и ПО с тем, чтобы корректно осуществлять таймирование и дальнейшее оценивание коэффициента ускорения. Ускорение зависит от общего времени выполнения программы. Обычно в параллельной программе присутствуют четыре фазы:

- ввод данных;
- вычисление параллельного алгоритма на p процессорах;
- обмен данными между p параллельными процессами;
- вывод данных.

Поэтому студент в каждом конкретном случае разрабатывает свою методику таймирования, которую защищает перед преподавателем. При этом следует обратить внимание на то, что таймирование можно осуществлять как с учетом времени на обмен данными по каналам связи между процессорами, так и без такого учета, а также с учетом или без времени на ввод-вывод данных. Сравнение ускорений полученных по различным методикам таймирования для одной и той же параллельной программы может дать полезную информацию для размышлений об эффективности разработанного ПО и путях ее повышения!

3. При применении способа 2 временных оценок параллельных программ следует иметь в виду, что при использовании данного критерия в качестве векторной операции можно брать либо простую векторную операцию, либо сложную «макрооперацию» (например, некоторый алгоритм). Важно, чтобы время выполнения этой векторной операции являлось функцией длины вектора.

Однако следует иметь в виду, что для получения пиковой производительности, которая бы выражалась, например, в общепринятых миллионах операций с плавающей точкой в секунду, необходимо ставить эксперимент, в котором происходит, например, сложение двух векторов различной длины. Тогда вычислив и построив соответствующий график зависимости производительности операции сложения от длины вектора, можно получить соответствующую оценку производительности именно в Mflops. В более сложных алгоритмах трудно получить непосредственные оценки пиковых производительностей, поскольку любая «макрооперация» состоит из определенного числа различных процессорных команд. Только зная их количество и времена выполнения каждой из них, можно попытаться решить задачу экспериментальной оценки максимального быстродействия ВС. Поскольку такой информацией при выполнении данной работы студент не располагает, то, оценивая r_∞, необходимо помнить, что единицей измерения будет «макрооперация» в секунду.

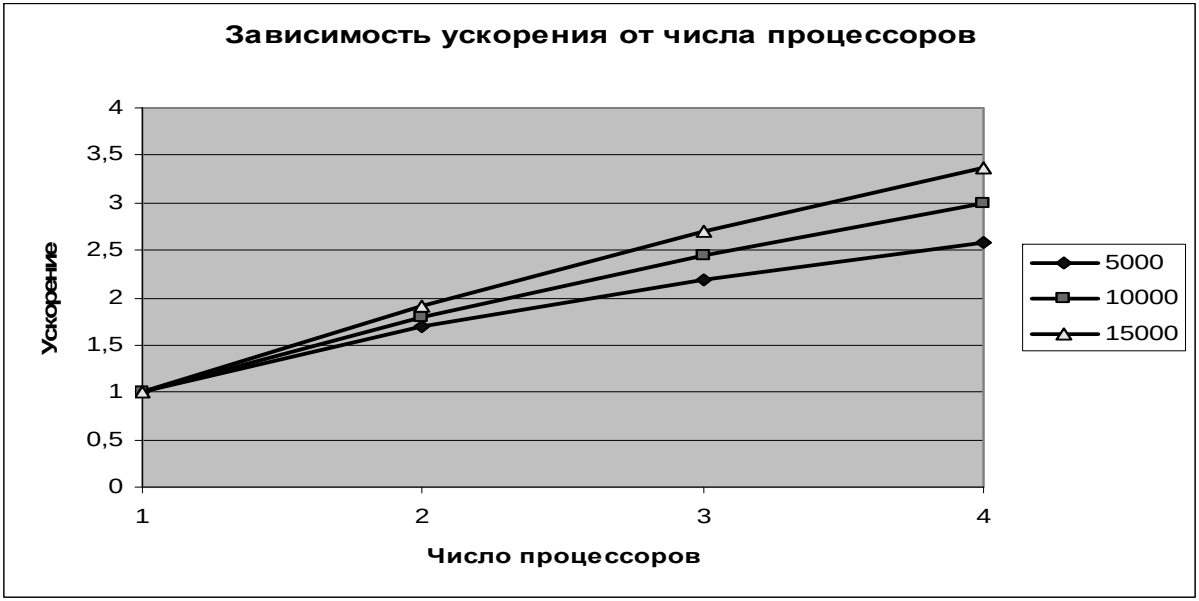

Рис. 3.7. Пример графика зависимости ускорения от числа процессоров

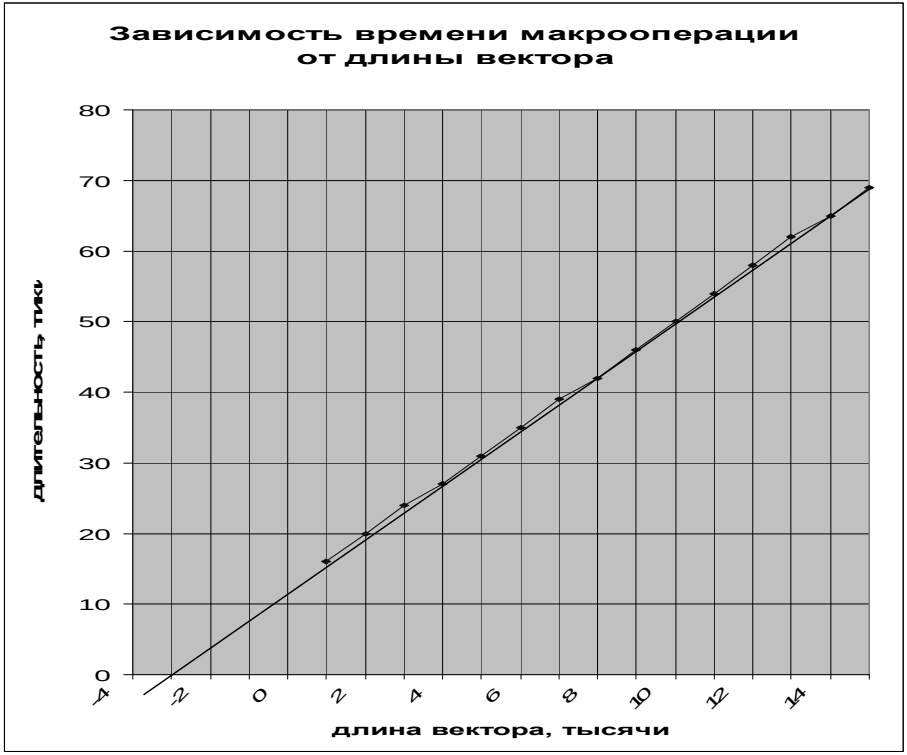

Рис. 3.8. Пример графика зависимости времени выполнения векторной макрооперации от длины вектора

4. Следует учитывать тот факт, что при таймировании время выполнения таймируемого участка программы измеряется в тиках, причем один тик равен 1 мс. Если значение таймирования получается равным нескольким десяткам или даже сотням тиков, то погрешности такого измерения значительны. Как уменьшить погрешность таких измерений? Самый простой способ – «зациклить» измеряемый участок, т.е. повторить вычисления несколько десятков или даже сотен раз, а затем поделить полученный результат на число циклов. Однако делать такое «зацикливание» надо корректно, при этом следует помнить, что любое задание снимается сервером по таймауту, равному 30 с.

На рис. 3.6 и 3.7 приведены примеры оформления результатов таймирования, проведенного в соответствии с некоторой методикой таймирования параллельной программы. Набор и количество графиков, которые необходимо построить в каждой конкретной работе определяются конкретной задачей, методикой таймирования, разработанной студентом, а также преподавателем.

5. В прил. 1 приведен пример программы, демонстрирующей технологии написания программы с обращением к библиотеке Veclib.

3.2.5. Задание по лабораторной работе 2

1. Разработать алгоритм и написать однопроцессорный вариант программы для полученной задачи.

2. Отладить написанную программу на Root-процессоре. Произвести таймирование однопроцессорной программы.

3. Проанализировать выданную задачу и выбрать рациональную конфигурацию (одну из четырех) соединений процессоров.

4. Разработать алгоритм параллельного выполнения для полученной задачи. Алгоритм должен обеспечивать рациональную (равномерную) загрузку процессоров.

5. Написать отдельные программные модули для каждого процессора.

6. Отладить параллельную программу в соответствии с инструкцией. Программа должна быть универсальной по отношению к входным данным.

7. Разработать и обосновать методику таймирования, положив в основу способ 1 или 2 (см. разд. 5.2.1) по согласованию с преподавателем.

8. Провести таймирование разработанной параллельной программы в соответствии с методикой п. 7. Построить соответствующий график. Для построения графиков использовать электронную таблицу типа Gnumeric в ОС Linux или Exel в ОС Windows.

9. Продемонстрировать и сдать преподавателю разработанные программы, методику таймирования и полученные в соответствии с ней результаты.

3.2.6. Варианты заданий к лабораторной работе 2

Вариант 1

Написать параллельную программу для вычисления определенного интеграла методом трапеций.

Подынтегральная функция: $F(x) = x$.

Интервал интегрирования: [0, 3].

Вариант 2

Написать параллельную программу для вычисления определенного интеграла методом трапеций с использованием библиотечной функции vtrapzf (см. прил. 3).

Подынтегральная функция: $F(x) = x$.

Интервал интегрирования: [0, 3].

Вариант 3

Написать параллельную программу для вычисления определенного интеграла методом Симпсона.

Подынтегральная функция: $F(x) = x$.

Интервал интегрирования: [0, 3].

Вариант 4

Написать параллельную программу для вычисления определенного интеграла методом Симпсона с использованием библиотечной функции vsimpsonf (см. прил. 3).

Подынтегральная функция: $F(x) = x$.

Интервал интегрирования: [0, 3].

Вариант 5

Написать параллельную программу для вычисления определенного интеграла методом трапеций.

Подынтегральная функция: $F(x) = \sin x$.

Интервал интегрирования: $[0, \pi]$.

Вариант 6

Написать параллельную программу для вычисления определенного интеграла методом трапеций с использованием библиотечной функции vtrapzf (см. прил. 3).

Подынтегральная функция: $F(x) = \sin x$.

Интервал интегрирования: $[0, \pi]$.

Вариант 7

Написать параллельную программу для вычисления определенного интеграла методом Симпсона.

Подынтегральная функция: $F(x) = \sin x$.

Интервал интегрирования: $[0, \pi]$.

Вариант 8

Написать параллельную программу для вычисления определенного интеграла методом Симпсона с использованием библиотечной функции vsimpsonf (см. прил. 3).

Подынтегральная функция: $F(x) = \sin x$.

Интервал интегрирования: $[0, \pi]$.

Вариант 9

Написать параллельную программу для вычисления определенного интеграла методом трапеций.

Подынтегральная функция: $F(x) = \tan x$.

Интервал интегрирования: $[0, 3\pi/8]$.

Вариант 10

Написать параллельную программу для вычисления определенного интеграла методом трапеций с использованием библиотечной функции vtrapzf (см. прил. 3).

Подынтегральная функция: $F(x) = \tan x$.

Интервал интегрирования: $[0, 3\pi/8]$.

Вариант 11

Написать параллельную программу для вычисления определенного интеграла методом Симпсона.

Подынтегральная функция: $F(x) = \tan x$.

Интервал интегрирования: $[0, 3\pi/8]$.

Вариант 12

Написать параллельную программу для вычисления определенного интеграла методом Симпсона с использованием библиотечной функции vsimpsonf (см. прил. 3).

Подынтегральная функция: $F(x) = \tan x$.

Интервал интегрирования: $[0, 3\pi/8]$.

Вариант 13

Написать параллельную программу для вычисления свертки одномерного сигнала в соответствии с выражением:

$$Y_i = \left(\sum (X_{i-k/2+j}) \times a_j \right) / (n+1),$$

где сумма вычисляется по индексу j. Здесь X_i – входной одномерный массив отсчетов, $i = 0, 1, 2, \ldots, n$; Y_i – выходной одномерный массив отсчетов, $i \in [k/2, (n - k/2)]$; a_j – коэффициенты свертки, $j = 0, 1, 2, \ldots, k$. Массив X_i – это массив отсчетов функции $\sin x$ на интервале $[0, 2\pi]$; $n = 100000$; $k = 10$.

Примечание. Значения n и k могут быть изменены по согласованию с преподавателем.

Вариант 14

Написать параллельную программу для вычисления преобразования Блэкмена – Харриса для одномерного сигнала в соответствии с выражением:

$$X_i = [0{,}35875 - 0{,}48829\cos(i2\pi / n) + 0{,}14128\cos(i4\pi / n) -$$

$$-0{,}01168\cos(i6\pi / n)]Y_i,$$

где $i \in [0, n - 1]$ – размерность массива данных; Y_i – входной одномерный массив отсчетов, $i = 0, 1, 2, \ldots, n - 1$; X_i – выходной одномерный массив отсчетов, $i = 0, 1, 2, \ldots, n - 1$. Рекомендуемое значение $n = 100000$. Значение n может быть изменено по согласованию с преподавателем.

Вариант 15

Написать параллельную программу для вычисления вектора как взвешенного среднего двух векторов:

$$X_i = (Y_i + \alpha X_i)/(\alpha + 1),$$

где X_i – входной и выходной одномерный массив (вектор) вещественных чисел, $i = 0, 1, 2, \ldots, n - 1$; Y_i – входной одномерный массив (вектор) вещественных чисел, $i = 0, 1, 2, \ldots, n - 1$. Рекомендуемое значение $n = 100000$. Значение α подбирается самостоятельно. Значение n может быть изменено по согласованию с преподавателем.

Вариант 16

Написать параллельную программу для вычисления вектора как взвешенного среднего двух векторов с использованием библиотечной функции vlinaveragesf:

$$X_i = (Y_i + \alpha X_i)/(\alpha + 1),$$

где X_i – входной и выходной одномерный массив (вектор) вещественных чисел, $i = 0, 1, 2, \ldots, n - 1$; Y_i – входной одномерный массив (вектор) вещественных чисел, $i = 0, 1, 2, \ldots, n - 1$. Рекомендуемое значение $n = 100000$. Значение α подбирается самостоятельно. Значение n может быть изменено по согласованию с преподавателем.

Вариант 17

1. Задан ориентированный взвешенный граф матрицей инцидентностей.
2. Найти кратчайший путь между *i*-й и *j*-й вершинами графа (алгоритм Дейкстры).

Вариант 18

1. Задан ориентированный взвешенный граф матрицей инцидентностей.
2. Найти кратчайшие пути между всеми парами вершин графа (алгоритм Флойда).

Вариант 19

1. Задан ориентированный взвешенный граф матрицей инцидентностей.
2. Найти кратчайшие пути между всеми парами вершин графа (алгоритм Данцига).

Вариант 20

1. Задан неориентированный взвешенный граф матрицей инцидентностей.
2. Построить минимальное покрывающее дерево графа (алгоритм Прима).

Вариант 21

1. Задан ориентированный взвешенный граф матрицей смежности.
2. Построить цикломатическую матрицу графа.

Примечания к вариантам 17 – 21.

1. Следует рассматривать только связные графы.
2. Число вершин графа $m < 20$.
3. Для вариантов задач на графах рекомендуется искать подробное описание алгоритмов в литературе [13, 14].

СПИСОК ЛИТЕРАТУРЫ

1. Levin V.K. Some questions on Development and Applications Transputers Systems in the USSR: British-Soviet Symposium on Transputers Systems. Moscow, 26 – 28/06/90.
2. *Аляутдинов Д.А., Далевич А.Н. Параллельный Си.: Уч. изд. М.: МАИ, 1991.*
3. Красковский А.Ю., Чепин Е.В. Транспьютеры: технические характеристики и опыт использования // Зарубежная радиоэлектроника. 1991. № 4. С. 43 – 50.
4. Транспьютеры. Архитектура и программное обеспечение: Пер. с англ. / Под ред. Г. Харпа. М.: Радио и связь, 1993.
5. *Хокни Р., Джессхоуп К. Параллельные ЭВМ. Архитектура, программирование и алгоритмы: Пер. с англ. М.: Радио и связь, 1986.*
6. Чепин Е.В. Структурная нотация современных микропроцессоров // Научная сессия МИФИ-99. Сб. научн. трудов. В 13 томах. Т. 7. М.: МИФИ, 1999. С. 99 – 100.
7. Parallel C. User Guide. Texas Instruments TMS320C40. 3L Ltd.
8. *Удаленный коллективный доступ к многопроцессорной системе / А.Б. Вавренюк Л.Д. Забродин В.В. Макаров А.Н. Никитин, Е.В. Чепин // Труды Всерос. научн. конф. «Высокопроизводительные вычисления и их приложения», г.Черноголовка, 2000. С. 72 – 73.*
9. Optimized DSP/Vector Library for TMS320C40/C30/. Reference Manual. Sinoctonalysis Inc., 1995.
10. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. СПб.: БХВ-Петербург, 2002.
11. Эндрюс Г.Р. Основы многопоточного, параллельного и распределенного программирования: Пер. с англ. М.: Изд. дом «Вильямс», 2003.
12. Configuration Guide for Sundance SMT320, SMT 320 User Guide.
13. Майника Э. Алгоритмы оптимизации на сетях и графах. М.: Мир, 1981.
14. Кристофидес Н. Теория графов. Алгоритмический подход. М.: Мир, 1978.
15. Рублев А.Н. Линейная алгебра. М.: Высшая школа, 1968.

Приложение 1

Пример программы на языке Параллельный Си: интегрирование методом Симпсона на одном процессоре с использованием библиотеки Veclib и технологии таймирования

Приложение состоит из одной программы h1. Задача h1 располагается на Root-процессоре. Текст приложения на языке Параллельный Си:

```
#include <stdio.h>
#include <chan.h>
#include <string.h>
#include <dspvec.h>
#include <timer.h>
#include <math.h>

/* Интегрирование методом Симпсона на 1 процессоре с использованием стандартной библиотеки Veclib */
/* отрезок [0.0 ;3.0], функция - синус*/

main(argc,argv,envp,in_ports,ins,out_ports,outs)
int argc,ins,outs;
char *argv[], *envp[];
CHAN *in_ports[], *out_ports[];

{
int i,ii,t1,t2;
double y[13000],x[13000];
int n;
float eps=0.001,integr,intlast;
double x0=0.0,xk=3.0;
double s,delta,step;

step=-0.50607;
s=-0.045;
s=fabs(s);
```

```
step=fabs(step);
printf("S= %f   step= %f\n",s,step);

printf("Simpson1 begins\n");
t1=timer_now();

for(ii=0;ii<10;ii++)
{
 n=500;
 intlast=12345.;
 do
 {
  integr=0.0;
  n=n*2;
  step=(xk-x0)/(n-1);
  x[0]=x0;
  for(i=1;i<n;i++)
  {
   x[i]=x[i-1]+step;
   y[i]=sin(x[i]);
  }
/*   vsin2sf(n,y,x,1,1);          */
  vsimpsonsf(n,step,y,x,1,1);
  integr=y[n];
  delta=intlast-integr;
  intlast=integr;
  printf("          delta=%f\n",delta);
 } while (fabs(delta)>eps);
}

t2=timer_now();

printf("TIME1*10= %d \n",(t2-t1));
printf("integr_summa=%f\n",integr);
}
```

Приложение 2

Библиотека некоторых функций Параллельного Си

Библиотека «The Parallel C Run Time Library» и «Optimized DSP/Vector Library for TMS320C40/C30» – набор компилируемых функций, которые обеспечивают выполнение операций, не включенных в язык Си. Некоторые из них приведены ниже в таблице. Функции, обеспечивающие коммуникацию по link-каналам, подробно описаны в основном тексте.

Эта библиотека содержит набор функций, которые могут выполняться в программе, но не включаются сами собой. Такие функции, прежде всего, двух типов: стандартный набор функций языка Си, а также специальные канальные функции, обслуживающие обмен информацией между процессорами. Для подключения соответствующей библиотеки функций необходимо использовать стандартный для Си оператор include <заголовок.h >. В табл. П.2.1 приведены некоторые из этих функций с указанием имени заголовочного файла.

Таблица П.2.1

№	Имя функции	Краткое описание	Обращение и типы аргументов (прототип)	Описание аргументов	Заголовочный файл
1	abs	Абсолютная величина целого аргумента	int abs(int arg)		<stdlib.h>
2	sin	Синус аргумента, заданного в радианах	double sin(double x)		<math.h>

Продолжение 1 табл. П.2.1

№	Имя функции	Краткое описание	Обращение и типы аргументов (прототип)	Описание аргументов	Заголовочный файл
3	cos	Косинус аргумента, заданного в радианах	double cos(double x)		<math.h>
4	tan	Тангенс аргумента, заданного в радианах	double tan(double x)		<math.h>
5	exp	Вычисление экспоненты	double exp(double x)		<math.h>
6	fabs	Абсолютная величина вещественного аргумента	double fabs(double arg)		<math.h>
7	fmod	Остаток от деления *x*/*y*	double fmod(double x, double y)		<math.h>
8	fprintf	Форматируемый вывод	Стандартные для языка Си		<stdio.h>
9	log	Вычисление логарифма	double log(double x)		<math.h>
10	log10	Вычисление десятичного логарифма	double log10(double x)		<math.h>
11	timer_now()	Возвращает значение таймера	int timer_now()	Разрешение таймера – одна миллисекунда, т.е. 1000 тиков за секунду	<timer.h>

Продолжение 2 табл. П.2.1

№	Имя функции	Краткое описание	Обращение и типы аргументов (прототип)	Описание аргументов	Заголовочный файл
12	vsin2sf	Синус вектора/массива	void vsin2sf(int n, float*x, float*y, intdx, intdy)	*n* – длина входного/выходного вектора; *x* – выходной вектор; *y* – входной вектор; *dx*, *dy* – шаг внутри векторов, обычно равен 1	<dspvec.h> *x*, *y* должны быть массивами!
13	vcos2sf	Косинус вектора/массива	void vsin2sf(int n, float*x, float*y, intdx, intdy)	*n* – длина входного/выходного вектора; *x* – выходной вектор; *y* – входной вектор; *dx*, *dy* – шаг внутри векторов, обычно равен 1	<dspvec.h> *x*, *y* должны быть массивами!
14	vtan2sf	Тангенс вектора/массива	void vsin2sf(int n, float*x, float*y, intdx, intdy)	*n* – длина входного/выходного вектора; *x* – выходной вектор; *y* – входной вектор; *dx*, *dy* – шаг внутри векторов, обычно равен 1	<dspvec.h> *x*, *y* должны быть массивами!

Окончание табл. П.2.1

№	Имя функции	Краткое описание	Обращение и типы аргументов (прототип)	Описание аргументов	Заголовочный файл
15	vsimpsonf	Вычисление интеграла методом Симпсона	vsimpsonsf(n,step,y,x,1,1) n-int step,x,y-float	*n* – размерность входного и выходного векторов; step – шаг интегрирования; *y* – выходной вектор частичных сумм; *x* – входной вектор; две единицы – значения шага по *x* и *y*, которые рекомендуется задавать = 1	<dspvec.h>
16	vtrapzf	Вычисление интеграла методом трапеций	vtrapzf(n,step,y,x,1,1) n-int step,x,y-float	*n* – размерность входного и выходного векторов; step – шаг интегрирования; *y* – выходной вектор частичных сумм; *x* – входной вектор; две единицы – значения шага по *x* и *y*, которые рекомендуется задавать = 1	<dspvec.h>
17	vlinavera-gesf	Вычисление взвешенного среднего двух векторов	vlinaver-agesf(n,alpha,x,y,dx,dy) int n,dx,dy float *x,*y float alpha	*n* – длина векторов; alpha – скалярная величина «взвешивания»; *x* – входной и выходной вектор; *y* – входной вектор; *dx* – шаг для *x*; *dy* – шаг для *y*	

А.Б. ВАВРЕНЮК, В.В. МАКАРОВ, Е.В. ЧЕПИН

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ПАРАЛЛЕЛЬНОМ СИ ДЛЯ МРР-СИСТЕМ

Лабораторный практикум

Под редакцией Е.В. Чепина

Учебное электронное издание

Редактор *М.В. Макарова*
Оригинал-макет изготовлен *М.В. Макаровой*

Подписано в печать 17.07.2008. Формат 60x84 1/16
Печ.л. 4,75. Уч.-изд.л. 4,75. Изд. № 022-1

Московский инженерно-физический институт
(государственный университет).
115409, Москва, Каширское ш., 31