Язык ознакомления с миром параллелизма

Л.В. Городняя

ИСИ СО РАН им. А.П. Ершова, НГУ

В докладе рассматривается проект языка обучения параллельному программированию Синхро, предназначенный для начального ознакомления с базовыми понятиями взаимодействия процессов и управления вычислениями.

Ключевые слова: параллельное программирование, обучения программированию, учебный язык программирования, реализационная прагматика, синхронизации процессов.

1. Введение

Возрастание актуальности обучения параллельному программированию требует развития языковой и системной поддержки введения в программирование [3]. В середине 1970-х годов активное исследование методов параллельного программирования рассматривалось как ведущее направление преодоления кризиса технологии программирования. Теперь рост интереса к параллельному программированию связан с сетевыми технологиями и переходом к массовому производству многоядерных архитектур. Прежде всего, следует упомянуть бурное развитие суперкомпьютеров и распределенных информационных систем, а также распространение многоядерных процессоров и многопроцессорных графических ускорителей [2].

Пришло время изучать информатику и программирование, начиная с мира параллелизма. Не исключено, что это даст и решение общих проблем практического программирования. Одна из таких проблем - выбор архитектуры абстрактного многопроцессорного настраиваемого макроассемблера для масштабируемой кодогенерации параллельных программ [7].

Обнаруженные в этой связи проблемы в значительной мере имеют образовательный характер. Для их решения предлагается специально разработанный язык ознакомления с явлениями и проблемами параллелизма, названный Синхро (Synchro), что отражает акцент внимания на изучение методов синхронизации процессов, порождаемых многопоточными программами. Описанию особенностей языка Синхро посвящен настоящий доклад. Предлагаемые решения апробированы на ряде конференций и семинаров по параллельным вычислениям в Абрау, Иркутске, Москве, Новосибирске, Перми, Томске, а также в рамках спецкурсов ММФ и ФИТ НГУ «Парадигмы программирования» и «Функциональное программирование» [3-6].

2. Выбор решений

Ниша параллельного программирования обременена резким повышением сложности отладки программ, вызванным комбинаторикой выполнения фрагментов асинхронных процессов. Если первый барьер к успеху в параллельном программировании обусловлен последовательным стилем мышления, то второй барьер зависит от не преодолённой трудоёмкости отладки программ. Хотя основная трудоёмкость жизненного цикла программ связана именно с тестированием и отладкой программ, это направление более чем за полвека профессионального программирования прогрессирует преимущественно за счёт «силовых» характеристик оборудования, концептуально почти не выходя за пределы возможностей экстракодов отладки программ в доязыковую эпоху [1].

Обычно учебные языки программирования обеспечивают быстрое изучение базовых понятий с получением первых успехов, что облегчает переход от абстрактной алгоритмизации к практике отладки программ на базе производственных систем программирования. При проектировании языка обучения параллельному программированию учтены идеи наиболее

известных учебных языков программирования, таких как Basic, Pascal, Logo, Grow, Karel и Робик, с целью учёта их образовательного потенциала, учебной специфики и типовых свойств реализационной прагматики. Особенно важен опыт методически обусловленного введения программистских понятий в языке начального обучения программированию Робик [6]. Тщательная проработка понятий в этом языке может служить базисом для любой профориентации учебного процесса по информатике, программированию и информационным технологиям. Ведущее понятие языка Робик - «исполнитель», причем исполнителей может быть много, и они могут обладать разными системами команд. Такое понимание допускает естественное развитие при переходе к представлению многопоточных программ и асинхронных процессов в терминах взаимодействия роботов, допускающего основные модели вычислений, опробованные в языках APL, SETL, SISAL (просачивание операций, семантика множеств, распараллеливаемые итерации).

Языки XXI века как правило поддерживают основные парадигмы программирования на языках высокого уровня – императивное, функциональное, логическое и объектно-ориентированное, что позволяет программировать решения задач с разным уровнем изученности в рамках единой языковой обстановки и получать навыки работы на всех фазах полного жизненного цикла программ в рамках одной системы программирования, что учтено в проекте языка Синхро. Интересно отметить, что в образовательной сфере наметилась тенденция причислять к принципиальным парадигмам программирования функциональное, параллельное и императивное, а логическое и объектно-ориентированное рассматривать как дополнительные парадигмы. Такая классификация возможно отражает зависимость парадигм от знания предметной области [5].

Переход к параллельным алгоритмам влечёт пересмотр содержания многих понятий и введение новых терминов, отражающих разного рода явления и эффекты, не имевшие особого значения для обычных последовательных алгоритмов. Программы становятся многопоточными, исполнение кода - многопроцессорным, действие может начать выполняться до завершения предыдущего, средства управления кроме логических значений используют сигналы и сообщения, обработка структур данных может требовать изменения порядка их обработки и многое другое. Возникающее в результате расширение пространства понятий меняет подходы к реализации решений, использующих параллелизм, и в некоторой мере сказывается собственно на постановках задач и планировании жизненного цикла программ решения задач, ориентированных на использование параллелизма [4]. Наиболее очевидные требования учебного уровня выглядят следующим образом:

– поддержка полного жизненного цикла программы решения заданной задачи, начиная от демонстрации отдельных примеров её входных данных и результатов до создания эффективного решения;

– пошаговое развитие программы по мере уточнения постановки решаемой задачи и созревания параллельного алгоритма её решения;

– лаконичное представление многопоточной программы, порождающей сложно устроеное взаимодействие процессов;

– удобство отладки программы от отдельных процессов до полной интеграции многопроцессорной конфигурации;

– внешнее управление программным экспериментом с помощью вспомогательных потоков без редактирования основного текста программы;

– смягчение требований к уровню абстрагирования при описании и изучении языка программирования.

3. Семантическая метафора

Описание языка Синхро использует метафору «фабрика разнотипных роботов для конструирования программно-управляемых игрушек». По этой метафоре конструктор строит игрушку как определённую обстановку из имеющихся деталей для комплекса взаимодействующих роботов. Роботы могут выполнять небольшой набор общих команд. Кроме того, каждый тип роботов обладает своей специальной системой команд и регистров, включая

шкалу сигналов о событиях и условиях, на которые робот может реагировать по ходу игры. Конкретный робот может иметь своё имя и свой сценарий поведения. При создании игрушки конструктор может такой сценарий уточнить. Можно заказывать изменение системы команд робота, реализуемое фабрикой как запуск производства новой серии роботов.

Постановка учебной задачи для её решения на языке Синхро формулируется в терминах управления роботами на основе сценариев, определяющих поведение комплекса роботов в заданной обстановке. Кроме обычных команд по изменению обстановки, объектов, статуса робота и меню допустимых команд, в языке Синхро имеются специальные команды для манипулирования роботами и обстановками, а также для горизонтальной и вертикальной синхронизации процессов в терминах событий, происходящих в разных процессах, возможно одновременно. При организации сложных данных используются общие структуры или средства композиции, такие как списки, вектора, множества и отношения «после», «взаимоисключено» и «одновременно».

Сценарии формируются как многопоточные программы, строящиеся из действий, выполнение которых может зависеть от условий готовности или вероятности срабатывания действий. Последнее означает, что ведется учет частоты выполнения действий как в системах для разработки компьютерных игр. Можно объявлять планируемую длительность выполнения сложных действий или пауз межу ними. Суммарно это даёт пространство процессов сравнимое с вариациями процессов в наиболее популярных схемах сетей Петри. Правило композиции влияет на упорядочение действий при их выполнении, возможно отличающееся от порядка вхождения в сценарий. Простейшие действия — это команды. Сложные действия строятся из более простых. При выполнении действия формируется сообщение о ходе процесса, т. е. начале (Н), собственно выполнении (В) и завершении (К) действия.

Обстановка устроена как Мета-Робот, схемно подобный роботу, команды которого выполняют другие роботы, включённые в игру и использующие детали обстановки. Детали могут встраиваться в роботы и выполнять роль регистров, образующих память робота. Регистры имеют фиксированное число состояний, переключение между которыми подчинено определённой дисциплине. Действия, связанные с обработкой памяти, подчинены механизму транзакций, т. е. признание действия безуспешными влечёт восстановление памяти в состояние, предшествующее этому действию. Специальные команды Мета-Робота позволяют воздействовать на включенных роботов и менять состояние их памяти. Роботы и регистры могут настраиваться на стартовое состояние до начала игры.

Все команды допускают безусловное и условное выполнение. Последнее предназначено для эффективного реагирования на события и условия. Условное выполнение команды приводит к сбросу соответствующего сигнала. Объявление события позволяет выполняться ждущим его действиям, обладающим готовностью. Игра управляется многопоточной программой, выполняемой комплексом роботов, включая Мета-Робота.

Каждый робот имеет встроенный самописец, регистрирующий ход процесса выполнения многопоточной программы. Простые роботы генерируют протокол, внося в него сообщения о командах. Безусловные команды последовательно вносят в протокол события <Н; В; К>, условные команды — <Н; В; К> или <Н; К> в зависимости от истинности ключевого сигнала. Мета-Робот вносит в протокол наименования роботов, объявляемые ими события и готовность к реагированию на события. Кроме того, собирается статистика частоты выбора вероятностных ветвей.

4. Управление процессами и структура данных

Семантическая метафора позволяет основные семантические системы языка, такие как вычисление, обработка памяти, структурирование данных и программ, управление процессами, представлять в виде определений комплекса взаимодействующих роботов. Синхро — язык низкого уровня, в нём не поддерживается иерархия определений фрагментов программы и структур данных.

Внешне можно писать программы и в императивном, и в функциональном стиле, но реализационная прагматика композиций действий допускает более широкое пространство допустимых процессов. Так например, для последовательного выполнения действий «А ; В»

возможны протоколы исполнения не только вида «Н-А; В-А; К-А; Н-В; В-В; К-В», но и содержащие вставки из параллельных процессов, что приводит протокол к виду «Н-А; П1; В-А; П2; К-А; П3; Н-В; П4; В-В; П5; К-В», где Пх — вставки протоколов действий из других процессов. Средства отладки учебных программ реализации небольших параллельных алгоритмов целенаправленно демонстрируют отклонения от непрерывности внешне императивных последовательностей, вызванные вставками фрагментов других параллельных процессов.

По умолчанию и теоретическим предпочтениям вычисление функциональных выражений ограничено критерием единственного чистого результата и отсутствием побочных эффектов. Но реализационная прагматика попутно формирует вспомогательные значения для обеспечения обратимости действий. При этом, порядок вычисления подвыражений может регулироваться синхронизацией в терминах барьеров. Кроме того, обработка структур данных допускает деструктивные воздействия при условии следования принципу сохранения элементов, что позволяет при необходимости восстанавливать исходные данные и выполнять транзакции памяти при неуспешных действиях. Обычные вычисления могут формировать более одного результата. Операции и функции допускают полиморфизм и автоматическое покомпонентное их распространение на структуры данных.

Память кроме механизма транзакций может быть подчинена разным дисциплинам функционирования. При обработке структур данных разделены порядок записи элементов, их вычислений и размещения в памяти. Управление допускает любые модели параллелизма, подобные разным схемам сетей Петри.

При вертикальной синхронизации объявление события позволяет выполняться ждущим его действиям, обладающим готовностью. Событие не сбрасывается, пока все ждущие его потоки не будут готовы или их выполнение не окажется невозможным. При горизонтальной синхронизации барьеры внутри циклов и рекурсий индексируются при копировании.

Программа – это комплекс из потоков, выполняющийся в общей памяти - заданном контексте, возможно с выделением значимых барьеров-синхронизаторов. Барьер устроен как логическая переменная, получающая значение «истина» при его достижении и «ложь» после его прохождения. Невыделенные барьеры не влияют на ход процесса. Должен существовать контекст, в котором синхронизация потоков корректна.

Поток – это комплекс из фрагментов, возможно синхронизованных по одноименным барьерам с другими потоками. Должен существовать контекст, в котором выполнимы все действия потока. Фрагменты могут быть устроены из асинхронных действий, порядок выполнения которых не задан, или как последовательность, задающая порядок выполнения действий порядком их вхождения. Кроме того, можно представлять ряд взаимно исключённых действий или аналоги ветвлений, включая цикл.

Возможность определять ветвящиеся процессы представлена в виде конструкций, подобных предохранителям или защищённым командам, позволяющим выполнять или не выполнять отдельные фрагменты в зависимости от условий, кратности или вероятности срабатывания действий.

ЕСЛИ Выр **ТО** Дир — по истинности выражения.
КРОМЕ Выр **ТО** Дир — исключая истинность выражения.
КОГДА — проверка условия пока оно не окажется истинным. Частота проверок не определена. Истинность условия не должна быть исключена.
ЖДУ — ожидание события.
[ВОЗМОЖНО] Выр **%** Дир — соответственно заданной вероятности.
ЦИКЛ … Дир — бесконечное итерирование, допускающее как внешнее, так и внутреннее ограничение.

Кроме семантики значений, предполагающей исключение побочных эффектов, поддерживается семантика реорганизации структур данных, ориентированная на «закон сохранения элементов», согласно которому поддерживается перемещение элементы из одних структур в другие при условии восстановимости исходных данных. Результат реорганизации – пара из полученной структуры и остатка преобразуемой, что удобно для программирования

фильтрации – нужное исчезает из аргумента, становящегося дополнением результата как остаток от фильтрации.

Все имена уникальны, т. е. нет локализации по блокам за исключением определений функций. Фактически это присваивания, но с сохранением старых значений, которое можно в любой момент достать специальными функциями.

При применении функции к ее аргументам поддерживается покомпонентное просачивание операций, фильтров и функций по любым структурам данных. Структура из функций дает структуру из результатов их применения к одним и тем же аргументам. Возможно превращение бинарной операции в фильтр, который строит пару из отфильтрованного и остатка. Структура из фильтров дает структуру из результатов их применения к одному и тому же аргументу.

Базовые структуры данных - списки, элементы которых доступны последовательно и размещены рассредоточено, и вектора с обычным индексированием элементов, размещенных в соседних регистрах. В том и другом случае последовательность вычисления элементов не обязана совпадать с последовательностью размещения. Разделитель «**;**» символизирует последовательность, «**,**» - произвольный порядок.

Структура = **(** (Выр [{ **;** | **,** }]) …**)**
| **[** [(Индекс…):] (Выр [{ **;** | **,** }])…**]**

Множества моделируются с помощью специальных операций.

Можно задать несколько исполнителей, работающих в общем мире. Взаимодействие процессов может регулироваться внешним образом с помощью синхронизации, что позволяет представлять бесконечные циклы и рекурсии без определения условий завершения в расчёте на использование разных счетчиков, контролирующих работу автомата. Синхронизовать действия счётчиков можно в виде отдельного потока.

При ознакомлении с миром параллелизма используются идеи олимпиадных задач и учебных примеров из школьных и факультативных курсов информатики. Отдельные приемы программирования на Синхро показаны ниже.

3 котлеты на 2 сковороды	
ОЧЕРЕДЬ жарить = (2 2 2), готово = () *%% в конце строки **;** не обязательна*	предстоит обжарить 2 стороны каждой котлеты тарелка для готовых котлет
ВЕКТОР сковороды [1..2]	
ПОКА жарить ЦИКЛ *%% цикл для любого числа сковород и котлет* *%% общая вертикаль выделяет блок* жарить → сковороды	**масштабируемый фрагмент:** до исчерпания недожаренных котлет
ЖДУ 5 сковороды = (сковороды - 1) *%% внутренний цикл:* ск [i] = ск [i] - 1,	два первых элемента из очереди исчезают, перемещаясь на сковороды время «жарки» значит, что одна сторона обжарена
сковороды (?≠ 0 **,** ?= 0) → (жарить, готово)	«недожаренные» (= 1) становятся в очередь на жарку, накапливается результат (со 2-го витка), разделили готовые и требующие жарки
РЕЗУЛЬТАТ готово *%% = (0 0 0)*	больше нечего жарить.

Пример 1. Олимпиадная задача для младших школьников «3 котлеты на 2 сковороды». Для обжарки одной стороны котлеты требуется 5 минут. Имеется 2 сковороды, на них надо как можно скорее поджарить 3 котлеты.

Примечание: На Intel-семинаре в Москве эта задача упоминалась как пример немасштабируемого алгоритма. Надо предложить и ясно записать масштабируемое решение.

```
ФУНКЦИЯ  QS (Data)
 [  ЕСЛИ  ( size (Data) < 2 TO Data ) ,
    КРОМЕ  ( size (Data) < 2
    (Pivot =  Data [ liml (Data)] ;
        Low, Mid, High  ← Data ((?<, ?=, ?<) Pivot )        %% просачивание по списку фильтров и структуре
   ; ( QS (Low) || Mid || QS (High) )
     )
  ]
```

Пример 2. Быстрая сортировка (часто приводится в описаниях языков параллелльного программирования).

числа A [1 .. 2K]; процессоры П [1 .. K] ; ЦИКЛ П [1 .. K] **!** %*%просачивание по вектору процессоров* (ФУНКЦИЯ (i) ЕСЛИ A [2*i - 1] < A [2*i] ТО A [2*i - 1] ⇔ A [2*i]} **;** *%% затем* ЕСЛИ A [2*i + 1] > A [2*i] ТО A [2*i] ⇔ A [2*i +1]) [1 .. K] *%%просачивание по индексам* ПОКА ВЫПОЛНЯЛОСЬ (⇔)	Вызов исполнителей ! Поведение одного процессора на окрестности числа (2*i): *ЕСЛИ* левое число меньше (2*i)-го, *ТО* они меняются местами. Затем процессор работает с правым от (2*i) числом: *ЕСЛИ* (2*i)-ое число меньше правого, *ТО* они меняются местами. Выход при отсутствии перестановок.

Пример 3. Параллельная сортировка (распараллеленный «пузырёк»).

5. Схема ознакомления с явлениями

Показать и пронаблюдать модели параллельных вычислений и связанные с их применением явления и проблемы можно на задачах, описанных в книге Хоара [8]. Решения представляются на языке Синхро, наследующем конструкции языков Logo, Karel, Grow и Робик с учетом современных технологий разработки информационных систем. Используется материал олимпиадных задач и учебных примеров из школьных и факультативных курсов информатики [3]. В целом порядок ознакомления может быть следующим:

- Роботы. Потоки выполнимых действий. Сценарии. Содержание деятельности. Готовность к выполнению. Выполнимость действий. Условия выполнения. Завершаемость процесса. Протокол. Игра в роботов.

- Миры исполнителей/роботов, контекст, память. Вспомогательные роботы. Ввод-вывод можно рассматривать как сценарии отдельных роботов, что удобно для чисто функционального программирования, поддерживающего логику деятельности без учёта ресурсов и времени, но допускающего горизонтальную синхронизацию, внешнее управление взаимодействиями, событиями, вводом-выводом и взаимоисключениями.

- Схемы управления действиями и фрагменты потоков. Рекурсия. Взаимодействия. Вертикальная синхронизация по времени. Будильник. Кратность действий. Транзакции.

- Взаимодействие потоков. События. Поломки и выигрыши при функционировании. Дозирование императивности исполнения и целостности данных. Императивно-процедурное программирование (диагностика, исключения, сбои. условия, императивность, целостность, транзакции, немедленное исполнение, время задержки действия).

- Серийная обработка данных/реквизита. Сценарии роботов можно рассматривать как грамматику допустимых процессов, представленных в виде протоколов.

- Автономная и попарная отладка потоков. Сопоставление протокола и сценария может

использоваться для локализации источников редких ошибок. Совмещение конечных и произвольных процессов.

- Функции и параметры. Локальный контекст. По множеству протоколов можно восстанавливать накрывающую их грамматику.
- Фильтры и рекурсия. Успех выполненного действия. Возможен подбор тестов, позволяющих сузить множество допустимых протоколов. Уточнение определений как предшественник основных идей объектно-ориентированного программирования (эволюция постановок задач, редактирование текстов сценариев, детализация задач и декомпозиция программ, интеграция сценариев).
- Реорганизация структур данных. Распределение циклов по процессорам. Копирование и пересылка данных. Независимость последовательности вычислений от порядка размещения результатов в векторах и потоках.
- Ленивое исполнение и накопление результатов. Визуализация протоколов может быть отдельным режимом работы системы программирования. Использование методов логического программирования (факты, макеты, правила, счёт, неполнота изученности).
- Тиражирование фрагментов и циклы. Грамматика протоколов позволяет наследовать технику эквивалентных преобразований.
- Преобразования многопоточных программ. Синхронизация процессов. Уточнение сценариев и сравнение результатов деятельности.

Особый круг образовательных проблем связан с навыками учёта особенностей многоуровневой памяти в многопроцессорных системах. Обычное последовательное программирование такие проблемы может не замечать, полагаясь на решения компилятора, располагающего статической информацией о типах используемых данных и способного при необходимости выполнить оптимизирующие преобразования программы. К сожалению, использование языков параллельного программирования в качестве языка представления программы не гарантирует её приспособленность к удачному распараллеливанию.

6. Заключение

Рассматривая задачу формализации языков параллельного программирования как путь к решению проблемы адаптации программ к различным особенностям используемых многопроцессорных комплексов и многоядерных процессоров, мы видим, что решение этой проблемы требует разработки новых методов компиляции программ с акцентом на тестирование, верификацию и отладку, а также, развития средств и методов ясного описания семантики языков параллельного программирования, включая представление программируемых преобразований текста и кода программы с удостоверением их корректности.

Разнообразие моделей параллельных вычислений и расширение спектра доступной архитектуры следует рассматривать как вызов разработчикам языков и систем программирования, решающим проблемы создания методов компиляции многопоточных программ для многопроцессорных конфигураций. Язык должен допускать представление любых моделей параллелизма, проявляемого на уровне решаемой задачи или реализуемого с помощью реальной архитектуры, причем такое представление требует разноуровневых форм и конструктивных построений, гарантирующих сохранение свойств программ при их реорганизации.

Предлагаемый язык Синхро представляет собой эксперимент по выбору базовых средств для достаточно полного решения проблем эффективной реализации параллельных алгоритмов, вынуждено требующих использовать весьма широкий спектр сложно совместимых средств от управляющих действий более низкого уровня, чем в привычных языков программирования, до манипулирования пространствами решений по обработке данных в памяти, типичных для языков сверх высокого уровня. Обзор исследований и решений в смежных областях представлен в [5,6]

Особый круг образовательных проблем связан с навыками учёта особенностей многоуровневой памяти в многопроцессорных системах. Обычное последовательное программирование такие проблемы может не замечать, полагаясь на решения компилятора,

располагающего статической информацией о типах используемых данных и способного при необходимости выполнить оптимизирующие преобразования программы. К сожалению, использование языков параллельного программирования в качестве языка представления программы не гарантирует её приспособленность к удачному распараллеливанию.

Литература

1. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Теория соответствия для систем с блокировками и разрушениями // М.: Физматлит, 2008. - 412 с.
2. Воеводин В. В., Воеводин Вл. В. Параллельные вычисления. // СПб.: БХВ-Петербург, 2002. — 608 с.
3. Городняя Л.В. О курсе «Начала параллелизма» // Ершовская конференция по информатике. Секция «Информатика образования». 27 июня - июля 2011 года. Новосибирск. с. 51-54.
4. Городняя Л.В. О проблеме автоматизации параллельного программирования // В сборнике Международной суперкомпьютерной конференции «Научный сервис в сети Интернет: многообразие суперкомпьютерных миров» — URL: http://agora.guru.ru/abrau2014 .
5. Городняя Л.В. Парадигмы программирования. Часть 4. Параллельное программирование //Новосибирск. Препринт ИСИ СО РАН. — URL: http://www.iis.nsk.su/files/preprints/gorodnyaya_175.pdf.
6. Городняя Л.В. Парадигмы программирования. Часть 5. Учебные языки программирования //Новосибирск. Препринт ИСИ СО РАН. — URL: http://www.iis.nsk.su/files/preprints/gorodnyaya_176.pdf
7. Звенигородский Г.А. Первые уроки программирования // Библиотечка Кванта, v.41. М.: Наука, 1985.
8. Степанов Г.Г. Пути обеспечения переносимости программ и опыт использования системы СИГМА // Трансляция и преобразование программ. – Новосибирск: ВЦ СО АН СССР, 1984. – 9 с.
9. Хоар Ч. Взаимодействующие последовательные процессы. // М.: Мир, 1989. 264 с.

Educational Programming Language for Parallel World

L.V. Gorodnyaya

A.P. Ershov Institute of Informatics Systems, Novosibirsk State University

The report is devoted to the language learning parallel programming named Synchro. It is language for initial study of basic concepts of the interaction processes and computing control.

Keywords: parallel programming, programming education, educational programming language, realizable pragmatics, process synchronization.

References

1. Burdonov I.B., Kosachev A.S., Kuljamin V.V. Teorija sootvetstvija dlja sistem s blokirovkami i razrushenijami [Theory of compliance for systems with locks and destruction] // M.: Fizmatlit, 2008. - 412 p

2. Voevodin V. V.,Voevodin Vl. V. Parallelnyje vychislenija [Parallel Computing] — SPb.: BHV-Peterburg, 2002. — 608 p.

3. Gorodnyaya L.V. O kurse «Nachala parallelizma» [On the Course "Principles of Parallelism "] // Ershov Informatics Conference . Workshop “Educational Informatics”. (27 ijunja - ijulja 2011). Novosibirsk. p. 51-54.

4. Gorodnyaya L.V. O probleme avtomatizacii parallelnogo programmirovanija [On the Problem of Automation of Parallel Programming] // In the book of the International Supercomputer Conference " Scientific Service on the Internet: the Variety of Supercomputing Worlds" — URL: http://agora.guru.ru/abrau2014 .

5. Gorodnyaya L.V. Paradigmy programmirovanija Chastj 4. Parallelnoje programmirovanie Programming Paradigms Part 4. Parallel Programming] //Novosibirsk . Preprint ISI CO RAN. — URL: http://www.iis.nsk.su/files/preprints/gorodnyaya_175.pdf .

6. Gorodnyaya L.V. Paradigmy programmirovanija Chastj 5. Uchebnyje jazyki programmirovanija [Programming Paradigms Part 5. Educational Programming Languages] //Novosibirsk . Preprint ISI CO RAN. — URL: http://www.iis.nsk.su/files/preprints/gorodnyaya_176.pdf

7. Zvenigorodskij G.A. Pervyje uroki programmirovanija [The First Programming Lessons] // Bibliotechka Kvanta, v.41. M.: Nauka, 1985.

8. Stepanov G.G. Puti obespechenija perenosimosti programm i opyt ispolzovanija sistemy SIGMA [Ways to programs mobility and experience of using the system SIGMA] // Translation and transformation of programs/ – Novosibirsk: VC SO AN SSSR, 1984. – 9 p.

9. Hoar Ch. Vzaimodejstvujuzhije posledovatelnyje processy [Communicating Sequential Processes.] //M.: Mir 1989. 264 p