УДК 004.43 (042.4)

ОРГАНИЗАЦИЯ ВЫЧИСЛЕНИЙ И РАБОТЫ С ПАМЯТЬЮ В УЧЕБНОМ ЯЗЫКЕ ПРОГРАММИРОВАНИЯ СИНХРО

Л. В. Городняя[1,2] **[0000-0002-4639-9032]**

[1]*Институт систем информатики им. А.П. Ершова СО РАН, 630090 г. Новосибирск, проспект Академика Лаврентьева, 6*

[2]*Новосибирский государственный университет, 630090, Новосибирск, ул. Пирогова, 1*

[1]gorod@iis.nsk.su

Аннотация

Статья посвящена ряду решений, принятых в проекте разрабатываемого в Лаборатории информационных систем Института систем информатики СО РАН учебного языка программирования СИНХРО, предназначенного для ознакомления с базовыми явлениями взаимодействия процессов и управления вычислениями над общей памятью. В центре внимания находится парадигма функционального программирования. Язык ориентирован на школьников младших и средних классов, а также студентов младших курсов и непрофессионалов. При обучении используется опыт оперирования игрушечными роботами, перемещающимися на клетчатой доске. Статья представляет интерес для всех, кто интересуется проблемами современной информатики, программирования и информационных технологий, особенно проблемами параллельных вычислений на суперкомпьютерах и распределённых системах, и вообще применением многопроцессорных комплексов.

Ключевые слова*: учебные языки программирования, виртуальная машина, система команд, функциональное программирование, восстановление данных, освобождение памяти, многопоточные программы, параллельные вычисления, общая память, взаимодействие процессов.*

ВВЕДЕНИЕ

Системе подготовки программистов предстоит освоить мир параллелизма и создать методику опережающего ознакомления с его не вполне очевидными явлениями, что и побудило к эксперименту по разработке учебного языка программирования СИНХРО [1]. Алгоритмы становятся параллельными, программы – многопоточными, компьютеры – многопроцессорными. Очередное действие может начать выполняться до завершения предыдущего, средства управления вычислениями кроме логических значений используют события и многое другое [2–6], что приводит к размежеванию синтаксиса и семантики параллельных вычислений на уровне выражений [7–9]. И всё это происходит в многоязыковой обстановке – мощность пространства языков программирования давно перевалила за десятки тысяч [10, 11].

Первая цель опережающего ознакомления с параллелизмом – профилактика жесткого привыкания к принципам самого распространённого однопроцессорного, последовательного, императивно-процедурного программирования, осложняющего овладение многопоточным программированием [12, 13]. Следует признать, что программирование и его техническая основа в последние десятилетия претерпели значительные изменения. Вторая цель – приобретение опыта применения механизмов, актуальность которых растёт в связи с усложнением ИТ и расширением классов сложных задач, обременённых трудно удостоверяемыми требованиями, такими как живучесть, надёжность и безопасность программных инструментов.

Изложение начинается с нестрогого описания основных механизмов языка СИНХРО (раздел 1) и обоснования решений по созданию и описанию учебного языка программирования языка СИНХРО (раздел 2). Затем описаны особенности семантики языка (раздел 3) и прагматики перехода от потоков к процессам (раздел 4), включая вытекающие из этого решения по системе команд виртуальной машины, допускающей поддержку взаимодействия процессов (раздел 5), и описание механизма взаимодействия общей и локальной памяти (раздел 6). В заключении даны выводы по образовательным проблемам параллельного программирования.

1. ОБЩИЕ ПОЛОЖЕНИЯ

Детское программирование давно уже не редкость [14–17], и при оценке образовательного значения парадигм программирования признают фундаментальными – функциональное, параллельное и императивно-процедурное, а логическое и ООП рассматривают как дополнительные парадигмы. Опережающее освоение параллелизма пока не получило признания, хотя реализация такого подхода возможна на базе учебных и новых языков программирования благодаря присущей им мультипарадигмальности.

Как это типично для учебных языков программирования, язык СИНХРО поддерживает как основные, так и фундаментальные парадигмы программирования, включая параллельное программирование и метапрограммирование. Использование разных парадигм приводит к использованию разных форм представления многопоточных программ, что методически обусловлено профилактикой привыкания к одной форме, которая может тормозить переход к новым формам (Эффект второй системы) [18]. Эти формы семантически эквивалентны, они порождают одно и то же семейство процессов.

Проект учебного языка СИНХРО рассчитан на формирование интуитивных моделей в процессе ознакомлении с основными явлениями параллельного программирования, встречающимися в учебно-методических и научных материалах, языках высокого уровня (ЯВУ), языках высокопроизводительного программирования и механизмах сетевых информационных сервисов, образующих инструменты современной ИТ-индустрии. Такие модели необходимы для успешного обучения в любой практической области, в частности, для перехода к производственной деятельности и обеспечению надёжности и безопасности ИТ.

Следует обратить внимание, что при обучении не всегда более простую сущность изучить легче, чем более сложную. Реально как простое воспринимается знакомое и привычное, а новое и непривычное кажется более сложным, даже если оно математически и конструктивно много проще.

Основные понятия языка СИНХРО – фрагменты выражений, действий, вычислений и данных, включая процессоры, программы, языки программирования или системы. В учебном процессе язык предстаёт как череда диалектов, соответствующих фазам обучения.

По умолчанию обстановка для выполнения программы независимо от диалекта содержит систему программирования на полном входном языке. Определение программы содержит контекст, в котором синхронизация набора потоков корректна, иначе программа выполняется до обнаружения невыполнимого действия и приостанавливается.

2. ВЫБОР РЕШЕНИЙ

Определение языка СИНХРО потребовало заметного числа неочевидных решений.

2.1. Учёт скрытой грамматики деятельности. Обучение параллельному программированию требует пересмотра методик обучения для формирования скрытой грамматики деятельности по организации процессов [19].

2.2. Выделение диалектов по целям обучения. Мульти – ознакомление с феноменами параллелизма; Трансформ – опыт метапрограммирования; Асинхр – приобретение навыков подготовки многопоточных программ; Синпар – изучение методов представления взаимодействующих процессов; АМК – эксперименты по многопроцессорному программированию. Диалекты можно использовать автономно и совместно.

2.3. Использование мультипарадигмальности. Стихийно сформированные интуитивные модели организации процессов недостаточно удобны при обучении функциональному, параллельному и логическому программированию потому, что в обычной жизни прецеденты применения таких парадигм встречаются не часто и обычно рассредоточено по времени – нет компактности [19].

2.4. Приоритет параллелизму. Предварительное освоение иллюстративного материала для целенаправленного формирования интуитивных моделей, поддерживающих парадигму деятельности по организации параллельных процессов [1, 5, 7, 12, 20].

2.5. Ведущая парадигма функционального программирования, освобождающая для начала от проблем с побочными эффектами и ошибок, сложно зависящих от реального времени.

2.6. Разнообразия архитектур. Многие идеи организации процессов на уровне операционных систем слишком ограничены механизмами параллельной обработке на базе очередей и векторов [21].

2.7. Методическая обусловленность введения понятий программирования. Учтены идеи известных учебных языков программирования (Basic, Pascal, Logo, Grow, Karel, Робик, A++, Oz). Акцент на опыт методики введения понятий программирования на базе языка начального обучения программированию Робик [13].

2.8. Взаимодействующие процессы. Энтони Хоар (Antony Hoare) давно отметил, что параллелизм – вызов интеллекту человека. В его книге «Взаимодействующие последовательные процессы» [5] показано, что параллельные композиции внешне не сложнее последовательных. Именно таким образом простота пользовательских интерфейсов маскирует истинную сложность программ и процессов вычислений.

2.9. Расширяемость языка и системы программирования. Многие понятия языка программирования при переходе к параллельному программированию претерпевают изменения, и возникает необходимость в дополнительных понятиях, расширяющих ядро языка на уровне оболочки или отдельной программы. При определении языка СИНХРО использованы идеи языков Lisp, F#, C#, допускающих динамическую обработку кода программ, дополненные гибридными решениями по обработке памяти.

2.10. Ленивые вычисления и мемоизация. Выполнение параллельных процессов часто требует независимости порядка вычислений от последовательности представления действий в программе. Отчасти это делается с помощью замыкания функций и мемоизации, сохраняющей соответствие между значениями параметров и полученных результатов.

2.11. Трансформационная семантика. Общие решения по обеспечению лаконизма, универсальности, конструктивности и расширяемости приводят к трансформационной семантике языка программирования. Это является естественным полигоном для метапрограммирования как основы для оптимизации программ, признаваемой как основная проблема разработки компиляторов [22].

2.12. Просачивание операций и функций. Результат операции над скалярами в языках параллельного программирования обычно автоматически распространен на произвольные однородные структуры данных. Этот механизм в языке СИНХРО распространяется на технику применения функций, что повышает лаконизм выражений [1, 23].

2.13. Фильтрация данных. Результат фильтрации исчезает из аргумента – он переносится в другую структуру данных [23–25], что удобно при подготовке программ при решении задач методом перебора.

2.14. Фрагменты программ и синтаксическое подобие. Для задания синтаксического подобия фрагментной переменной её значениям используется понятие вида переменной. При определении макросов бинарная операция «~~» задаёт вид левого аргумента с помощью строки, являющейся правым аргументом. Все вхождения фрагментной переменной в схему программы должны быть синтаксически эквивалентны её вхождению в строку, задающую вид фрагментной переменной [26, 27].

2.15. Метапрограммирование для преобразования программ. В случае многопоточных программ возможно преобразование потоков, нацеленное на сведение к однородной системе. Обратимость преобразований и чувствительность их результата к информационным связям между фрагментами программы даёт умение формализовать критерии применимости трансформаций и выбора подходящего варианта [22].

2.16. Взаимодействие локальной и общей памяти. Предложена гибридная реализация императивной синхронизации взаимодействия локальной и общей памяти, совмещающая возможности «сборки мусора» и счетчика указателей на элементы памяти. [21, 28, 29].

2.17. Клетчатое поле. Дано предпочтения задачам, допускающим иллюстрацию решений в терминах перемещения по клетчатой доске [13].

2.18. Относительное описание языка программирования. Описание языка представляет собой череду диалектов, каждый из которых определён как дополнение к предыдущему диалекту.

3 МНОГОПОТОЧНЫЕ ПРОГРАММЫ

«Не верь глазам своим!»

Конкретный свод решений по семантике языка СИНХРО связан с классом рассматриваемых многопоточных программ.

3.1. Локализация имен используется лишь при определении исполнителей, потоков и функций. Нет иерархии и статической типизации.

3.2. Многопоточная программа представляется как набор потоков действий и может быть размечена барьерами на слои. Можно задавать внутреннюю и внешнюю очереди барьеров, определяющие порядок выполнения помеченных барьерами слоёв и действий [5].

3.3. Достижение барьера рассматривается как событие. Событие не сбрасывается, пока все ждущие его потоки не будут готовы или их выполнение не будет оценено как невозможное. [6].

3.3. Выражения могут использовать размещённые в общей памяти данные, но не изменяют их. Директивы могут изменять хранимые в общей памяти данные [1, 28, 29].

3.4. Структуры данных в программе наполняются элементами, порядок вычисления которых может отличаться от порядка вхождения в программу [28, 29]. Роль такого синтаксического отделения показана на примерах в таблице 1. Различие проявляется в мощности семейства допустимых процессов вычисления, что можно видеть на примерах (табл. 2), демонстрирующих такую разницу на простых выражениях со списками и векторами, воспринимаемыми зрительно как равнозначные, содержащие одинаковые элементы, но семантически они различны.

В первом примере согласно порядка записи в программе вычисляется a’, затем b’. После этого в памяти размещается ↓b, получается список (b), затем в этом списке размещается ↓a и получается список (a b). Во втором примере b’ может быть вычислено раньше, чем a’, и его результат может быть сразу или позже размещён ↓b в списке (b) до вычисления a’ и размещения полученного результата ↓a. Это даст такой же список (a b). Выражения «(a, b)» и «(a; b)» функционально эквивалентны, а мощность пространств допустимых процессов вычислений для них отличается.

Таблица 1. Порядок вычислений не зависит от порядка доступа к данным

№	Выражение	Пояснение
1	'('Выр (';' Выр)…')'	Список последовательно вычисляемых элементов, заполняемый в порядке записи в программе.
2	'('Выр (',' Выр)…')'	Список, заполняемый элементами в порядке записи в программе, вычисляемыми асинхронно, возможно в другой последовательности.
3	'['Выр (',' Выр)…']'	Вектор, заполняемый элементами в порядке записи в программе, вычисляемыми асинхронно, возможно в другой последовательности.
4	'['Выр(';’ Выр)…']'	Вектор, заполняемый последовательно вычисляемыми элементами в порядке записи в программе.

В третьем примере размещение 1:↓a первым элементом вектора может произойти как раньше вычисления b’, так и позже, результат которого размещается вторым элементом вектора 2:↓b, а размещение 1:↓a может быть выполнено после размещения 2:↓b. В четвёртом примере допускается возможность вычисления b’ раньше вычисления a’. Результат совпадает с результатом третьего примера, различна лишь мощность пространств допустимых процессов вычисления. Выражения «[a; b]» и «[a, b]» функционально эквивалентны. При ясном понимании теоретической разницы, на лабораторной практике многие студенты учитывают допустимые процессы лишь первого и третьего примеров, глаза воспринимают текст как последовательность независимо от разделителя. В задачу разработки языка СИНХРО входит поддержка лабораторных работ для формирования навыков учёта полных семейств допустимых процессов. Многопоточная программа допускает асинхронное выполнение потоков и действий, что означает в соответствии с принципом равноправия параметров функций независимость порядка вычисления выражений от порядка их вхождения в структуры данных, рассматриваемые как функции над этими выражениями.

Таблица 2. Разница в числе допустимых процессов вычисления выражений с разными структурами данных

№	Выражение	Процессы вычисления	Пояснение
1	(a; b)	{ a’ b’ ↓b ↓a }	Один допустимый процесс
2	(a, b)	{ a’ b’ ↓b ↓a | b’ a’ ↓b ↓a | b’ ↓b a’ ↓a }	Три варианта допустимых процессов
3	[a; b]	{ a’ b’ 2:↓b 1:↓a | a’ 1:↓a b’ 2:↓b | a’ b’ 1:↓a 2:↓b }	Три варианта допустимых процессов
4	[a, b]	{ a’ b’ 2:↓b 1:↓a | a’ 1:↓a b’ 2:↓b | a’ b’ 1:↓a 2:↓b | b’ 2:↓b a’ 1:↓a | b’ a’ 1:↓a 2:↓b | b’ a’ 2:↓b 1:↓a }	Шесть вариантов допустимых процессов

где x’ – вычисление выражения x, ↓x – размещение полученного значения x в памяти, n:↓x – размещение значения x как элемента вектора с номером n, { … }– множество допустимых процессов, | – разделитель элементов множества процессов.

3.5. Навыки предвидеть проблемы. В задачи ознакомления с параллелизмом входит научиться обнаруживать и предвидеть опасности, перестраивать и отлаживать программы при обнаружении неудачных взаимодействий потоков. Директива при благополучном исходе обработки памяти даёт результат подобно выражению.

3.6. Условия выполнения действий. При отладке сценариев можно задавать ориентировочное время срабатывания отдельных действий. Условное выполнение команды приводит к сбросу соответствующего сигнала. Действия, связанные с изменением состояния памяти, подчинены механизму транзакций, т. е. признание их безуспешными влечёт восстановление памяти в состояние, предшествующее этому действию.

3.7. Определение исполнителя выглядит как сценарий его функционирования, запускаемый при вызове исполнителя. При исполнении сценария ведется

учет частоты выполнения вероятностных действий как в системах для разработки компьютерных игр.

3.8. Постановка учебной задачи для экспериментов с параллельными алгоритмами на языке СИНХРО формулируется в терминах оперирования исполнителями [13].

3.9. Действия допускают схемы обхода, выглядящие подобно оператору IF без ELSE.

3.10. Потоки могут отлаживаться независимо друг от друга в предположении, что каждый из них можно располагать в отдельном контексте, их могут выполнять разные процессоры. Кроме того, приняты следующее ограничения:

- взаимодействия потоков выполняются только через синхронизацию;
- одновременно исполняемые потоки могут обмениваться данными через общую память;
- поддерживается восстановление многократно выполняемых фрагментов.

4. ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ: ОТ ПРЕДСТАВЛЕНИЯ ПОТОКОВ К ОРГАНИЗАЦИИ ПРОЦЕССОВ

При ознакомлении с параллелизмом полезно несколько расширить свод базовых понятий программирования, чтобы допускать разнообразие возможных учебных задач, позволяющих сформировать интуитивную картину многопоточности. В однопроцессорном программировании нет необходимости в представлении процессора – он существует по умолчанию всегда. Во многих учебных задачах для ознакомления с параллелизмом в роли процессоров выступают разные роботы, учебно-игровые исполнители, программируемые устройства, многопроцессорные конфигурации, поэтому представление процессоров или выполняемых ими процессов становятся столь же необходимым данным как представления значений и функций. Можно вспомнить, что в UNIX унифицировано представление файлов, устройств, заданий и процессов.

В языке РОБИК использовался образ фабрики роботов, которых можно было придумывать, тиражировать, именовать, выключать или включать в обстановку по мере развития сюжета учебной игры или оперирования программой. В языке mpC используются номера процессоров. Сети управления взаимосвязанными по-

токами – не более чем усложнённая структура данных, допускающая равноправное расположение данных, действий и процессоров в узлах сети, примерно, как это проектировалось в языке БАРС. Система управления выполнением действий учитывает геометрию сети, допускающую синхронизацию и преобразования фрагментов программ при их сборке, отладке и оптимизации, что образует полигон для учебных задач по мета программированию. На так расширенном своде понятий можно предлагать новые учебные задачи, включая оттеснение маловероятных вычислений, балансировку нагрузки процессоров, представление пространства итераций. Возможны эксперименты по автоматизированному и ручному распараллеливанию программ, моделированию необходимой при отладке идентичности повторных прогонов на суперкомпьютерах и представлению много контактных узлов для достижения эффективного использования нестандартных аппаратных решений.

Пришло время признать, что при переходе к производственным программам и параллельным вычислениям удобство приложения, работоспособность и производительность программ на практике оцениваются выше, чем полнота отладки и повышение эффективности, мало заметные пользователю. Корректность параллельных программ часто наследуется от ранее созданных последовательных программ решения тех же самых задач. Параллелизм привлекается лишь ради ускорения вычислений. Массово применяется изрядное число жизнеспособных или работоспособных программных систем, корректность которых сомнительна, что нисколько не мешает их популярности. Известны и примеры резкого взлёта удобных языков программирования, существенно проигрывающих в эффективности привычному семейству C/C++/C#.

На практике работает иное понимание правильности, обусловленное ожиданиями или привычками пользователя. Оно не формализовано и не вполне согласуется с общепринятыми и научно обоснованными рекомендациями, нацеливающими на эффективность или производителльность, а также на формальную верификацию, хотя верификация заметно прогрессирует в рамках парадигмы функционального программирования и со временем может смягчить груз ответственности программиста за качество программных продуктов. Возникает специ-

альная задача обеспечивать успех и производительность программ без потери достоинств формальной верификации и системных решений, нередко дающих улучшение, значительно превосходящее теоретические прогнозы.

Обычно в системы производственного функционального программирования включают механизмы практичного компромисса, внешне выглядящие как специальные функции, не нарушающие стиль записи программ. Например, Lisp 1.5 [30], Clisp, Cmucl, Clojure и другие члены семейства Lisp предоставляют варианты функций, обладающих разными свойствами при сохранении функциональной эквивалентности. В их числе контроль типов данных, схемы циклов, возможность восстановления данных, программируемые прогнозы времени счёта и объёма расходуемой памяти, мемоизация, обеспечивающая многократное использование результатов вычислений, и псевдофункции, дающие доступ к разным устройствам, начиная со средств ввода-вывода и обработки файлов.

По умолчанию виртуальная машина языка СИНХРО имеет хотя бы один процессор, на котором происходит выполнение основной программы. Компилятор по многопоточной программе, представленной средствами высокого уровня, строит функционально эквивалентную ей многопроцессорную программу низкого уровня, выполнимую на виртуальной машине. Комплект потоков при компиляции преобразуется в комплекс процессов, каждый из которых обычно выполняется одним процессором, если не возникает проблем неравномерной загрузки процессоров. Система команд виртуальной машины выбрана из соображений удобства конструирования шаблонов компиляции функций и операций языка СИНХРО. Предполагается, что ознакомление с параллелизмом включает упражнения на конструирование шаблонов, ориентированных на разные многопроцессорные конфигурации.

Контекст выполнения программы при переходе к процессам становится общей памятью, данные из неё доступны всем процессам многопроцессорной программы. Общая память содержит представления имён и значений глобальных переменных, возможно изменяемых разными процессами с помощью так называемых «деструктивных» функций, работающих подобно привычным присваиваниям. Различие в том, что каждая деструктивная функция обладает чистым функциональным эквивалентом, позволяющим отладку выполнять в два шага. Сначала

безопасная отладка под защитой принципа неизменяемости данных, затем – повышение эффективности выборочной заменой чистых функций на их деструктивные аналоги. Память состоит из двух частей – блока регистров прямого доступа к глобальным переменным и кучи произвольных структур данных для хранения значений переменных с протоколами, содержащими старые значения после присваивания на случай необходимости восстановления состояний памяти.

Для простоты изложения здесь не рассматривается учет разнообразия категорий систем команд отдельных процессоров и видов используемой памяти с различной дисциплиной функционирования, хотя в учебных задачах всё это имеет место. Механизмы освобождения памяти обычно требуют приостановки вычислений, что снижает общую производительность программы. Это особенно заметно при необходимости освобождать общую память, приостанавливая все процессы комплекса. Возможные подходы к смягчению этой проблемы рассмотрены в разделе 6.

5. ВИРТУАЛЬНАЯ МАШИНА АМК

При функционировании виртуальной машины, в отличие от реальной, число процессоров может изменяться, что влечёт появление динамических запросов к памяти, не всегда заметных при статическом анализе и компиляции программы. Кроме обычных процессоров в выполнении программы могут участвовать дополнительные устройства, которые можно рассматривать как специальные процессоры, способные выполнять разные действия по запросам обычного процессора. Каждый обычный процессор выполняет один процесс, порождаемый одним рядом команд. Шаги разных процессов не синхронизованы, но могут происходить одновременно.

Учитывая особенности компиляции языков, поддерживающих парадигму функционального программирования, при описании системы команд ядра языка СИНХРО используем расширение машины SECD, предложенной П. Ландиным для спецификации аппаратной реализации языка Lisp. Машина работает над регистрами, приспособленными к хранению списков. При определении виртуальной машины языка СИНХРО к регистрам системы команд «SECD» добавлен регистр «m» (Memory) для описания команд над общей памятью. Состояние процесса

полностью определяется содержимым этих регистров. Система команд виртуальной машины поддерживает выполнение результата компиляции функций и операций языка СИНХРО. Часть команд определена в книге П. Хендерсона [31], другие описаны в статье [21], здесь показаны лишь те, что нужны для иллюстрации взаимодействия потоков и работы с общей памятью.

M(SECD)+ – обозначение многопроцессорного комплекса над общей памятью, оно символизирует, что M – общая память для всех процессоров, а (SECD)+ – что хотя бы один процессор обязателен, общее число процессоров произвольно, и не исключено изменение их числа в динамике.

При описании шагов процесса используются следующие обозначения:

s – Stack – стек для окончательных и промежуточных результатов,

e – Environment – контекст для размещения локальных значений переменных,

c – Control_list – управляющая вычислениями программа,

d – Dump – резервная память для обеспечения надёжности вычислений.

m – Memory – общая память, выглядит как внешний регистр, доступный всем процессам.

m s e c d → m' s' e' c' d' – шаг процесса

– переход от старого состояния к новому.

m s e c d → **{** m' s' e' c' d' **|** … **}**

– переход, изменяющий состояние группы процессов.

{ m s e c d **|** …**}** → m' s' e' c' d'

– переход, зависящий от группы процессов.

AM = m s e c d

– именование текущего состояния процесса на одном процессоре.

AMk – имя отдельного процесса AM с номером k.

– Процесс знает свой номер «k».

Шаги работы виртуальной машины происходят как изменение состояния процесса или изменение состояний группы процессов. Возможны переходы, требующие учёта состояний группы процессов.

Для четкого отделения обрабатываемых данных от остальной части списка в описания команд используются обозначения, как в книге П. Хендерсона:

(x . L) – это список, в котором первый, головной элемент списка – x, а остальные, хвостовые находятся в списке L;

(x y . L) – в таком списке первый элемент – x, второй элемент – y, остальные находятся в списке L и т. д.

Для поддержки параллельных вычислений требуется несколько команд по организации комплексов процессов (неупорядоченных наборов – KIT) и рядов действий (последовательностей – ROW) с передачей их результатов и явной обработкой ошибок. Процессы образуются следующими командами:

KIT – комплекс из процессов для выполнения рядов действий, построенных в результате компиляции комплекта потоков;

ROW – ряд действий, созданный при компиляции потока как один процесс;

REZ – результат процесса и его номер размещаются в стеке;

WAIT – ожидание приостановки процесса (завершения или сообщения);

SEND – сообщение и его тэг передаются указанному процессу;

NEXT – ожидание полного завершения предыдущего действия процесса.

```
AM0 = m (2 1 . s) e (KIT c1 c2 . c) d
      → { AM0 = m s e c d
        | AM1 = @m s e c1 d
        | AM2 = @m s e c2 d
        }
```

Порядок порождения процессов произвольный. Процесс AM0 работает как монитор на основном процессоре и существует по умолчанию. В данном примере в произвольном порядке создано два процесса AM1 и AM2, для каждого из которых порождён свой процессор. Общая память для порождённых процессоров доступна через ссылку.

Введено обозначение:

@x – адрес позиции «x» или доступ к общей памяти в целом из процессора.

Процесс выполняется как ряд действий. Виртуальная машина поочерёдно

1 Для простоты иллюстраций рассматривается два процесса, а в процессе два действия.

запускает действия ряда, выделяя каждому из них свой процессор. Запуски действий происходят в порядке записи в программе, а завершения действий могут происходить в произвольном порядке, что соответствует множеству допустимых процессов, показанному на примерах раздела 1.

```
AMt = m (2 . s) e (ROW c1 c2 . c) d
        → { AMt = m (1 . s) e   ( ROW c2 . c) d
            | AM1 = @m (1 . s) e ( c1 SEND 1 AMt) d
            }
```

Процесс AMt запускает из двух предстоящих действий первое – c1, создавая для него свой процесс AM1, и уменьшает счётчик предстоящих действий ряда на 1. В стеке процесса стоит его номер. Созданный процесс AM1 сообщит процессу AMt о завершении выполнения действия c1.

```
AMt = m (1 . s) e ( ROW c2 . c) d
        → { AMt = m s e c d
            | AM2 = @m (2 . s) e (c2 ( SEND 2 AMt)) d
           }
```

Потом, когда-нибудь, процесс AMt запустит второе действие – c2 и для него создаст другой процесс AM2. Уже оставался 1 элемент – выход из рекурсии порождения процессов [2] для выполнения ряда действий. Завершилось ли первое действие – пока не известно. Если нужна строгая очерёдность действий, её надо при компиляции обеспечивать сообщениями или командой NEXT.

По мере завершения ряда действий процесса выполняется сборка результата командой REZ, выполняющей запись в стек число результатов комплекса процессов или результаты действий с номерами потоков.

```
{ AMt = m s e c d                    % c2 уже запущен,
| AM1 = @m (a 1 . s) e (REZ) d
```

2 Здесь в процессе два действия

```
  % получен результат c1 для передачи в AMt, возможно
  % требующий свёртки для строгого результата.
 | AM2 = @m (2 . s) e (WAIT AM1 c2) d
  % c2 дожидается завершения c1
 } % переход по готовности трёх процессов

          → { AMt = m (1 a . s) e (ass 2. c) d
                % AMt забирает результат c1 и
                % ждёт свёртки с результатом c2.
              | AM2 = @m (2 . s) e c2 d
      % c2 становится исполнимым фрагментом
            }
```

ass – представление функции, выполняющей свёртку указанного числа элементов в стеке и выдающей одно данное, рассматриваемое как строгий результат. Например: список, структура, сумма, произведение, макс, мин, последний и т. п. Со временем и второй процесс будет завершён, что позволит основному процессу выполнить свёртку результатов этих двух процессов.

6. НЕИЗМЕНЯЕМОСТЬ И/ИЛИ ВОССТАНОВЛЕНИЕ ДАННЫХ

Локальные переменные, хранимые в памяти обычных процессов, подчинены принципу неизменяемости, а глобальные переменные в общей памяти доступны через блок прямого доступа, для них вместо неизменяемости поддержан механизм восстановления оттеснённых присваиваниями значений. Наличие структуро сохраняющих чисто функциональных эквивалентов для деструктивных функций гарантирует гладкий спуск от строгого функционального программирования в направлении дозированной императивности. Для возможности восстанавливать значения переменных они сопровождаются протоколом, в котором новые значения размещаются вслед за именем переменной, а к прежним данным можно вернуться при необходимости, например, при отладке посмотреть историю изменения данных.

При выполнении обратимых действий допускается и обратимость воздействий на общую память, хранящую переменные с неизменяемыми указателями

на изменяемое значение. Каждый элемент общей памяти в любой момент времени обрабатывается только в одном процессе программы, подобно захвату-освобождению файла или устройства. В протоколе изменений можно видеть какой процесс изменил значение данной переменной и, при необходимости, вернуть утраченное значение.

Для описания команд работы с общей памятью используются дополнительные обозначения:

[n] – содержимое n-го элемента общей памяти.

r=[…] – установка значения регистра виртуальной машины.

m=**[**. . . x … y . . .**]** – можно указывать отдельные элементы памяти.

[… Var**:** Val … **]** – размеченное множество, содержащее элемент Val , помеченный меткой Var.

Регистр «m» представляет собой размеченное множество, метки в котором являются именами переменных и допускают прямой доступ, а помеченные элементы расположены в куче, содержат значения переменных и протоколы их изменения, выглядящие как последовательность имён процессов с установленными этими процессами значениями.

Если память «m» содержит для переменной X хранимое в памяти значение xt, установленное процессом AMt, то хранящийся в памяти элемент имеет вид «X: (xt AMt xt AM1 x1 AM2 x2 . . .)», где помеченный именем переменной X список содержит её текущее значение xt, а далее размещены имена процессов и ранее установленные этими процессами значения переменной. Здесь показано, что переменная X ранее имела значения x1 и x2, установленные процессами AM1 и AM2 соответственно, а теперь процесс AMt установил ей значение xt. Если после этого процесс AMk присвоит переменной X значение xk, то после такого изменения элемент памяти приобретёт вид

m'=[. . . X : (xk AMk xk AMt xt AM1 x1 AM2 x2 . . .)].

Произойдёт побочный эффект присваивания, допускающий при необходимости восстановление прежних значений.

Воздействия на общую память сводятся к пересылкам и обменам данными:

LD – размещение в стек «s» константы из регистра управления «C».

MLL – пересылка головного элемента из одного списка в другой;

MLV– пересылка головного элемента списка в элемент вектора;

CHNG – обмен данными в общей памяти комплекса.

При пересылке элемент из списка исчезает, а появляется в другой структуре данных. Команды пересылок и обмена данными в общей памяти рассматриваются как средства низкого уровня для профилактики возникновения временных интервалов между парами взаимосвязанных воздействий на память на уровне аппаратуры.

Для примера рассмотрим подробнее команду CHNG – обмен данными в общей памяти. При компиляции выражения «X <=> Y», представляющего в потоке R обмен значениями переменных X и Y, где X=x и Y=y, в скомпилированном для него коде процесса AR будет выстроен список команд вида «(LD Y CHNG X . c)». Это значит, что сначала произойдёт загрузка имени переменной Y в стек текущего процесса, затем будет обмен значениями с переменной X из регистра управления. Обмен значениями происходит в общей памяти, если в ней уже расположены значения этих переменных. Получается, что при организации обмена компилятор размещает в стеке имя одной переменной, а в регистре управления программой – имя другой переменной.

AR = m s e (LD Y CHNG X . c) d

→ m (Y . s) e (CHNG X . c) d

% Сначала команда LD подготовит первого участника

Пусть выполнена команда LD и память имеет состояние, содержащее значения переменных X и Y, а Px и Py – протоколы с прежними значениями этих переменных соответственно:

m=[. . . X: (x . Px) . . . Y: (y . Py) . . .]

Это условие возможности выполнять команду обмена:

AR = m=[. . . X: (x . Px) . . .

Y: (y . Py) . . .] (Y . s) e (CHNG X . c) d

→ m'=[. . . X: (y AR y . Px) . . .

Y: (x AR x . Py) . . .] ((y x) . s) e c d)

Прежние протоколы Px и Py для переменных X и Y остались без изменений, но в новых протоколах будет (X=y, а Y= x) с указанием, что это сделал процесс AR.

Здесь процесс AR производит обмен значениями x и y между переменными X и Y, в результате обмена обе переменные получат новые значения, они в протоколах сопровождены именем процесса AR, выполнявшего обмен, между действиями обмена стороннее вмешательство невозможно. Сформирован строгий результат из новых значений переменных X и Y – (y x) .

7. МОДЕЛИ ВЗАИМОДЕЙСТВИЯ ОБЩЕЙ И ЛОКАЛЬНОЙ ПАМЯТИ

Освобождение общей памяти может использовать решения, подобные опробованным в практике операционных систем при организации управления распределёнными системами с совмещением работы независимых программ. Обычно такие решения используют специальные структуры данных. Здесь рассмотрим варианты использования паспорта процесса и выделения пассивного процессора общей памяти.

Допустим, что каждый локальный процесс вместо доступа к общей памяти обладает вектором с именами или адресами используемых в нём глобальных переменных, что можно рассматривать как паспорт процесса, работающий как размазанная переменная. При освобождении локальной памяти процесса формируется новое, возможно уменьшенное, состояние этого вектора, доступное в любой момент и процессору общей памяти. Обычные процессы вместо полного доступа к общей памяти «m» в таком случае обладают лишь вектором «v» для доступа к определённым регистрам из блока прямого доступа. Вместо формата «m s e c d» тогда используется формат «v(m) s e c d», где «v(m)» – регистр для паспорта процесса, использующего общую память «m».

v(m) s e c d → v'(m') s' e' c' d' – переход от старого состояния виртуальной машины к новому, где «v» – вектор доступа к глобальным переменным из общей памяти «m».

Обычные процессы несколько изменяют метод взаимодействия с общей памятью. Она уже не принадлежит им равноправно, а доступна через паспорт – вектор указателей на данные в общей памяти. Различие можно показать на примере команды MLL – пересылка из головного элемента одного списка в другой список, где этот элемент становится головным. Списки доступны через указатели x и y для переменных X и Y соответственно.

Пусть паспорт процесса АМ содержит указатели на значения двух переменных X и Y. Это значит, что в общей памяти «m» сохраняются соответствующие этим переменным списки со значениями x и y, ранее установленные возможно другими процессами.

```
v (m)=[X: @x Y: @y …]
m=[X: @x … Y: @y … X: (x AMi (xh . xt) . Px) …
                        Y: (y AMj (yh . yt) . Py)
    …  ]
```

По указателям на значения этих переменных x и y в памяти M доступны данные «x=(xh . xt)» и «y=(yh . yt)»:

```
m=[x=(xh . xt) y=(yh . yt)
  % значения переменных X и Y до обмена
    X: @x  Y: @y                % паспорт процесса АМ
X: (x AMi (xh . xt) . Px) Y: (y AMj (yh . yt) . Py)
  % данные переменных X и Y
   … ]
```

Новое состояние памяти должно приобрести вид

```
m'=[x=xt y=(xh yh . yt)      % значения после обмена
     X: @x Y: @y             % паспорт без изменений
     X: ( x AM x AMi (xh. xt) . Px )
     Y: ( y AM y AMj (yh . yt) . Py)
 % данные о переменных изменены
 % при неизменном указателе
% указан АМ, изменивший значения по указателю
   ...]
```

В таком случае возможно выполнение команды MML для переменных X и Y.

v(m) (X . s) e (MLL Y . c) d → v(m') (X Y . s) e c d

Или более подробно:

v(m=[x=(xh . xt) y=(yh . yt) X: @x Y: @y
X: (x AMi (xh . xt) . Px)
Y: (y AMj (yh . yt) . Py] …) (X . s) e (MLL Y . c) d
→ v(m'=[x= xt y=(xh yh . yt) X: @x Y: @y
X: (*x* AM *x* AMi (xh. xt) . Px)
Y: (*y* AM *y* AMj (yh . yt) . Py)
...]) (X Y . s) e c d

Изменилось состояние памяти «m», доступное по прежним адресам изменённых значений *x* и *y,* и пополнились протоколы оттеснённых значений переменных X и Y. Паспорт не изменился, но он теперь ссылается на изменённое состояние общей памяти.

Такой подход при освобождении общей памяти даёт возможность частичного освобождения памяти, не задействованной в объединении паспортов локальных процессов, без приостановки всех процессоров.

Несколько проще модель со специальным процессором общей памяти, поддерживающим пассивный процесс, память которого размечена на занятые и свободные регистры, а команды выполняются лишь по запросам из активных процессов. В таком случае многопроцессорный комплекс состоит из обычных процессоров и одного процессора общей памяти, с которым могут взаимодействовать обычные процессоры. Между собой процессоры взаимодействуют только через общую память. Каждый обычный процессор может выполнять активный процесс из ряда команд, среди которых встречаются команды запросов к пассивному процессу общей памяти. Это обычные команды, выполняемые по ходу активного процесса без особенностей, управляющие доступом к общей памяти, всеми обменами данных и пересылками значений переменных, упомянутыми в разделе 5.

Команды запроса активных процессов к общей памяти имеют двойников среди команд пассивного процесса общей памяти. Это команды типа ленивых вычислений, возбуждаемые при выполнении запросов из активных процессов. Ко-

манды пассивного процесса общей памяти структурированы в ряд очередей, каждая из которых соответствует активному процессу и содержит двойников запросов в том же порядке, что и в активном процессе. Пара запрос из активного процесса и его двойник из пассивного процесса здесь названа «дублет». Она выполняются неразделимо, одновременно в стиле рандеву языка Ada. Механизм рандеву обеспечивает исключение случайного вмешательства сторонних процессов в работу общей памяти. (см. табл. 3).

Таблица 3. Дублеты – парные команды – срабатывают только неразрывно вместе

Запросы	*Двойники запросов*	*Неразделимая пара*	*Примечание*
GIVE	*TAKE*	GIVE-TAKE	Запрос на переменную
SAFE	WRITE	SAFE-WRITE	Запись значения переменной
READ	*VAR*	READ-VAR	Запрос значения переменной
UNDO	UNDO	UNDO-UNDO	Восстановить прежнее значение переменной
FREE	*FREE*	FREE-FREE	Освободить память от переменной

H – Имя переменной для данного в общей памяти. Все имена различны, они известны во всех процессах, определены при компиляции.

@H – адрес текущего значения переменной H в регистре, хранящем и протокол изменения значений – историю присваиваний.

<@H … > – шкала свободных регистров, содержащая адрес @H.

K – кратность доступа к переменной в текущий момент – число локальных процессов.

H*K – имя переменной с указанным числом использующих её процессов.

((v s e (C1 . c1) d | v s e C2 . c2) d **))**– одновременное срабатывание первых команд C1 и C2 из регистров «c» двух процессов.

Для примера рассмотрим команду-дублет GIVE-TAKE. При её выполнении на стеке процесса s имя переменной H замещается на адрес её значения @H, а имя переменной заносится в паспорт процесса v. Одновременно @H адрес в общей памяти исчезает из шкалы свободных регистров.

```
((v (H . s) e (GIVE . c) d → (H . v) (@H . s) e c d
            % адрес для значения в стеке
 | m=[<@H … > … ] s e (TAKE . c) d → m'=[ … H*1 ...] s e c d
   % новая переменная, счётчик был установлен в 0, теперь он = 1.
 ))% команды GIVE и TAKE срабатывают только одновременно
% переменная H уже была,
% ей уже выделена память другим процессом.
 ((v (H . s) e (GIVE . c) d → (H . v) (@H . s) e c d
 | m=[… (H*K): Ph …] s e (TAKE . c) d
   →m'=[… (H*(K+1)): Ph …] s e c d
 ))
% команды GIVE и TAKE срабатывают одновременно
```

Описанная схема работы с памятью позволяет объединять методы освобождения памяти типа «сборки мусора» с методами счётчиков доступа при условии исключения слияния ссылок.

ЗАКЛЮЧЕНИЕ

В статье описаны решения, принятые в проекте разрабатываемого в Лаборатории информационных систем ИСИ СО РАН учебного языка программирования СИНХРО, предназначенного для ознакомления с параллелизмом и особенностями многопоточного программирования, а также приобретения навыков организации взаимодействия процессов над общей памятью и профилактики страха перед побочными эффектами при определении сценариев отладки учебных программ. Главные из них – поддержка независимости порядка вычислений от порядка записи выражений в программе в соответствии с принципом независимости параметров, расширение понятия «данное» на представление процессоров и процессов, а понятия «структуры данных» – на сети взаимодействующих потоков

в соответствии с принципом универсальности, учёт динамики критериев при переходе от одной парадигмы программирования к другой в соответствии с принципом гибкости ограничений, описание системы команд виртуальной машины для определения учебного языка параллельного программирования в соответствии с принципом само-применимости. Кроме того, предложены две модели работы с общей памятью, позволяющие от неизменяемости данных перейти к методам восстановления данных. Такие решения смягчают противопоставление методов строго функционального программирования, нацеленного на правильность программ, и императивно-процедурного программирования, позволяющего привычными способами повышать эффективность и производительность программ.

Рассматривая задачу использования учебных языков параллельного программирования как путь к решению проблемы адаптации методов вычислений к различным особенностям используемых многопроцессорных комплексов и многоядерных процессоров, следует видеть, что решение этой проблемы, кроме ознакомления с проблемами и методами, требует разработки новых методик обучения и методов реализации программ с акцентом на тестирование, верификацию и отладку, а также развития средств и методов ясного описания семантики языков параллельного программирования, включая средства метапрограммирования для представление программируемых преобразований текста и переносимого кода программы с удостоверением их корректности.

Предлагаемый язык СИНХРО представляет собой эксперимент по выбору базовых средств для достаточно полного решения проблем эффективной реализации параллельных алгоритмов, вынужденно требующих использовать весьма широкий спектр сложно совместимых средств: от управляющих действий более низкого уровня, чем в привычных языках программирования, до манипулирования пространствами решений по обработке данных в памяти, типичных для языков сверхвысокого уровня. Обзор исследований в смежных областях представлен в работах [32, 33].

Особый круг образовательных проблем связан с навыками учёта особенностей многоуровневой памяти в многопроцессорных системах. Обычное 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. Новые и долгоживущие языки программирования, как правило, имеют мультипарадигмальный характер, что приводит к идее их расширения на многие модели параллельных вычислений.

Новые языки программирования содержат разнообразные конструкции для представления программ параллельных вычислений. Нужна система обучения, приспособленная не только к ознакомлению с отдельными эффектами и средствами параллельных вычислений, но и поддерживающая осознание особенностей взаимодействия процессов, удобная для экспериментов по формированию интуитивных навыков подготовки работоспособных программ и понимания новых возможностей аппаратуры. Не исключено, что этому противостоит исторически сложившаяся традиция одномерного, линейного представления текстов программ в виде бесконечной строки. В данной статье рассмотрен ряд форм, которые могут быть полезны при решении учебных задач по организации взаимодействующих параллельных процессов, включая процессы над общей памятью, на уровне решений, предложенных в языке учебного программирования СИНХРО.

Во введении определены цели опережающего ознакомления с параллелизмом. В нестрогом описании основных механизмов языка СИНХРО отмечена его структура как череда диалектов и расширенная трактовка ряда понятий программирования (раздел 1). Сформулирован ряд решений по созданию языка СИНХРО, включая учёт скрытой грамматики деятельности в процессе обучения, эксперимент по относительному описанию диалектов языка программирования и гибридных механизмов взаимодействие локальной и общей памяти (раздел 2). При описании ряда особенностей семантики языка особо показано различие в мощности семейства допустимых процессов вычисления для функционально эквивалентных семантически различных конструкций (раздел 3). Показана прагматики перехода от потоков к процессам, включая вытекающие из этого решения по системе команд, обеспечивающие неделимость действий по обмену данными (раздел 4), а также, дополняющие неизменяемость данных возможностью их восстановления (раздел 5) и допускающие механизм взаимодействия разных методов освобождения памяти (раздел 6).

Благодарность. Благодарю Дмитрия Владимировича Мажугу за экспериментальную реализацию ядра языка СИНХРО на языке функционального программирования Clojure.

СПИСОК ЛИТЕРАТУРЫ

1. *Городняя Л.В.* Язык параллельного программирования Синхро, предназначенный для обучения. Новосибирск: ИСИ им. А.П. Ершова СО РАН, 2016. 30 с. (Препринт/ИСИ СО РАН; № 180).
URL: https://www.iis.nsk.su/files/preprints/gorodnyaya_180.pdf.

2. *Городняя Л.В.* О курсе «Начала параллелизма» // Ершовская конференция по информатике. Секция «Информатика образования». 27 июня 2011 года. Новосибирск: ИСИ им. А.П. Ершова СО РАН. С. 51–54.

3. *Пеппер П., Экснер Ю., Зюдхольд М.* Функциональный поход к разработке программ с развитым параллелизмом // Системная информатика. Вып 4. Методы теоретического и системного программирования. Новосибирск: Наука. Сиб. изд. фирма, 1995. С. 334–360.

4. *Воеводин В.В., Воеводин Вл.*В. Параллельные вычисления. СПб.: БХВ-Петербург, 2002. 608 с.

5. *Хоар Ч*. Взаимодействующие последовательные процессы. М.: Мир, 1989. 264 с.

6. *Быстров А.В.* Сетевые средства синхронизации процессов // Тр. Всесоюзного научно-технического семинара «Программное обеспечение многопроцессорных систем». Калинин, 1985. С. 44–46.

7. *Городняя Л.В.* Парадигмы программирования. Часть 4. Параллельное программирование. Новосибирск, 2015. 74 с. (Препр. / ИСИ СО РАН; № 175). URL: https://www.iis.nsk.su/files/preprints/gorodnyaya_175.pdf.

8. *Лавров С.С.* Расширяемость языков. Подходы и практика. В сб.: Прикладная информатика, вып. 2. М.: Финансы и статистика, 1984. С. 17–22.

9. *Городняя Л.В.* О неявной мультипарадигмальности параллельного программирования. Материалы конференции «Научный сервис в сети Интернет: труды XXIII Всероссийской научной конференции». С. 104–116.
URL: https://library.keldysh.ru/prep_vw.asp?pid=9203.

10. *Городняя Л.В.* Методика парадигмального анализа языков и систем программирования // Научный сервис в сети Интернет: труды XXI Всероссийской научной конференции (23–28 сентября 2019 г., г. Новороссийск). М.: ИПМ им. М.В. Келдыша, 2019. С. 262–277.
URL: http://keldysh.ru/abrau/2019/theses/03.pdf doi:10.20948/abrau-2019-03.

11. *Андреева Т.А.* и др. Компьютерные языки как форма и средство представления, порождения и анализа научных и профессиональных знаний // Тр. XV Всерос. научно-методической конф. «Телематика–2008». СПб., 2008. С. 77–78.

12. *Городняя Л.В.* Парадигмы программирования. Ч. 5. Учебные языки программирования. Новосибирск, 2015. 60 с. (Препр. / ИСИ СО РАН; № 176).
URL: http://www.iis.nsk.su/files/preprints/gorodnyaya_176.pdf.

13. *Звенигородский Г.А.* Первые уроки программирования. Библиотечка «Кванта». М.: Наука, 1985. Т. 41. URL: http://cip.iis.nsk.su/files/course/zven-ves.pdf.

14. *Кушниренко А.Г., Грибанова И.Н., Райко М.В., Зайдельман Я.Н.* Как мы проводим вводное занятие по алгоритмике в разновозрастной группе дошкольников и младших школьников. URL:
https://fgoskomplekt.ru/upload/iblock/a2c/4o0ce5xyjyrha1b8qwsgxlhnjqp1iqme.pdf.

15. Что такое Лого. URL: http://www.int-edu.ru/logo/logo.html.

16. *Голиков Д.В., Голиков А.Д.* Книга юных программистов на Scratch – SmashWords, 2013. ISBN 978-1310227554.

17. *Ершов А.П.* Программирование – вторая грамотность. URL:
http://ershov.iis.nsk.su/ru/second_literacy/article.

18. *Брукс Ф.* Мифический человеко-месяц, или Как создаются программные системы. СПб.: Питер, 2021. 368 с.

19. *Городняя Л.В.* О функциональном программировании // Компьютерные инструменты в образовании (в печати).

20. *Городняя Л.В.* О проблеме автоматизации параллельного программирования // Научный сервис в сети Интернет: многообразие суперкомпьютерных миров: Труды Международной суперкомпьютерной конференции (22–27 сентября 2014 г., г. Новороссийск). М.: Изд-во МГУ, 2014. С. 191–196.
URL: http://agora.guru.ru/abrau2014

21. *Городняя Л.В.* Абстрактная машина языка программирования учебного назначения СИНХРО // Вестник Новосибирского государственного университета. Серия: Информационные технологии. 2021. № 4. С. 16–35.

22. *Адамович А.И., Климов Анд.В.* Об опыте использования среды метапрограммирования Eclipse/TMF для конструирования специализированных языков // Научный сервис в сети Интернет: труды XVIII Всероссийской научной конференции (19–24 сентября 2016 г. г. Новороссийск). М.: ИПМ им. М.В. Келдыша, 2016. С. 3–8. URL: https://doi.org/10.20948/abrau-2016-45.

23. *Городняя Л.В.* Подходы к представлению синтаксиса языков программирования. Новосибирск, 2019. 39 с. (Препр. / ИСИ СО РАН; № 185).
URL: https://www.iis.nsk.su/files/preprints/Preprint_185.pdf

24. About SETL and GNU SETL. URL: https://setl.org/doc/setl.html.

25. *Schwartz Jacob T*. Set Theory as a Language for Program Specification and Programming. Courant Institute of Mathematical Sciences, New York University, 1970.

26. *Kondratyev D.A.*, *Promsky A.V.* Developing a self-applicable verification system. Theory and practice // Automatic Control and Computer Sciences. 2015. Vol. 49. Issue 7. P. 445–452. URL: https://doi.org/10.3103/S0146411615070123

27. *Малышкин В.Э.* Технология фрагментированного программирования // Параллельные вычислительные технологии (ПаВТ’2012): труды международной научной конференции (Новосибирск, 26–30 марта 2012 г.). Челябинск: Издательский центр ЮУрГУ, 2012. С. 598–605.
URL: http://omega.sp.susu.ru/books/conference/PaVT2012/short/212.pdf

28. *Городняя Л.В.* Модели работы с памятью в учебном языке программирования СИНХРО // Научный сервис в сети Интернет: труды XXIV Всероссийской научной конференции (19–22 сентября 2022 г., онлайн). М.: ИПМ им. М.В. Келдыша, 2022. С. 137–154.

29. *Городняя Л.В.* Работа с данными в учебном языке программирования СИНХРО // Суперкомпьютерные дни в России: Труды международной конференции. 26–27 сентября 2022 г., Москва / Под. ред. Вл. В. Воеводина. М.: МАКС Пресс, 2022. ISBN 978-5-317-06875-2 e-ISBN 978-5-317-06876-9
https://doi.org/10.29003/m3109.RussianSCDays2022. С. 87–97.
https://doi.org/10.29003/m3109.RussianSCDays2022(внешняя ссылка)

30. *McCarthy J.* LISP 1.5 Programming Mannual. The MIT Press., Cambridge,

1963. 106 p.

31. *Хендерсон П.* Функциональное программирование. М.: Мир, 1983. 349 с.

32. *Городняя Л.В.* Гуманитарные факторы программирования /Сиб. Отделение Рос. Акад. наук, Ин-т систем информатики им. А.П. Ершова. Новосибирск: Изд-во СО РАН, 2020. 163 с.

33. *Городняя Л.В.* Функциональное программирование. Парадигма, модели, методы / Сиб. Отделение Рос. Акад. наук, Ин-т систем информатики им. А.П. Ершова. Новосибирск: Изд-во СО РАН, 2022. 482 с.

ORGANIZATION OF CALCULATIONS AND WORK WITH MEMORY IN THE EDUCATIONAL PROGRAMMING LANGUAGE SYNHRO

L. V. Gorodnyaya [1, 2] **[0000-0002-4639-9032]**

[1] *A.P. Ershov Institute of Informatics Systems SB RAS, 6, Acad. Lavrentjev pr., Novosibirsk 630090, Russia*

[2] *Novosibirsk State University, 1, st. Pirogova, Novosibirsk, 630090, Russia*

Abstract

The article is devoted to a number of decisions made in the project of the educational programming language Synchro, which is being developed at the Laboratory of Information Systems of the *A.P. Ershov IIS SB RAS*, designed to familiarize with the basic phenomena of the interaction of processes and control of calculations over shared memory. The focus is on the functional programming paradigm. The language is aimed at schoolchildren of primary and secondary grades, as well as junior students and non-professionals. During training, the experience of operating with toy robots moving on a checkered board is used. The article is of interest to everyone who is interested in the problems of modern computer science, programming and information technology, especially the problems of parallel computing on supercomputers and distributed systems, and in general the use of multiprocessor systems.

Keywords*: educational programming languages, virtual machine, command system, functional programming, data recovery, memory release, multithreaded programs, parallel computing, shared memory, process interaction.*

REFERENCES

1. *Gorodnyaya L.V.* YAzyk parallelnogo programmirovaniya Sinkhro, prednaznachennyy dlya obucheniya. Novosibirsk: ISI im. A.P. Yershova SO RAN, 2016. 30 s. (Preprint/ISI SO RAN; № 180).
URL: https://www.iis.nsk.su/files/preprints/gorodnyaya_180.pdf.

2. *Gorodnyaya L.V.* O kurse «Nachala parallelizma» // Yershovskaya konferentsiya po informatike. Sektsiya «Informatika obrazovaniya». 27 iyunya 2011 goda. Novosibirsk: ISI im. A.P. Yershova SO RAN. S. 51–54.

3. *Pepper P., Eksner YU., Zyudkhol'd M.* Funktsional'nyy pokhod k razrabotke programm s razvitym parallelizmom // Sistemnaya informatika. Vyp 4. Metody teoreticheskogo i sistemnogo programmirovaniya. Novosibirsk: Nauka. Sib. izd. firma, 1995. S. 334–360.

4. *Voyevodin V.V., Voyevodin Vl.V.* Parallel'nyye vychisleniya. SPb.: BKHV-Peterburg, 2002. 608 s.

5. *Khoar CH.* Vzaimodeystvuyushchiye posledovatel'nyye protsessy. M.: Mir, 1989. 264 s.

6. *Bystrov A.V.* Setevyye sredstva sinkhronizatsii protsessov // Tr. Vsesoyuznogo nauchno-tekhnicheskogo seminara "Programmnoye obespecheniye mnogoprotsessornykh sistem". Kalinin, 1985. S. 44–46.

7. *Gorodnyaya L.V.* Paradigmy programmirovaniya. Chast' 4. Parallel'noye programmirovaniye. Novosibirsk, 2015. 74 s. (Prepr. / ISI SO RAN; № 175).
URL: https://www.iis.nsk.su/files/preprints/gorodnyaya_175.pdf.

8. *Lavrov S.S.* Rasshiryayemost' yazykov. Podkhody i praktika. V sb.: Prikladnaya informatika, vyp. 2. M.: Finansy i statistika, 1984. S. 17–22.

9. *Gorodnyaya L.V.* O neyavnoy mul'tiparadigmal'nosti parallel'nogo programmirovaniya Material konferentsii "Nauchnyy servis v seti Internet: trudy XXIII Vserossiyskoy nauchnoy konferentsii. S. 104–116.
URL: https://library.keldysh.ru/prep_vw.asp?pid=9203.

10. *Gorodnyaya L.V.* Metodika paradigmal'nogo analiza yazykov i sistem programmirovaniya // Nauchnyy servis v seti Internet: trudy XXI Vserossiyskoy nauchnoy konferentsii (23–28 sentyabrya 2019 g., g. Novorossiysk). M.: IPM im. M.V. Keldysha, 2019. S. 262–277. URL: http://keldysh.ru/abrau/2019/theses/03.pdf URL: https://doi.org/10.20948/abrau-2019-03.

11. *Andreyeva T.A. i dr.* Komp'yuternyye yazyki kak forma i sredstvo predstavleniya, porozhdeniya i analiza nauchnykh i professional'nykh znaniy // Tr. XV Vseros. nauchno-metodicheskoy konf. «Telematika-2008». SPb., 2008. S. 77–78.

12. *Gorodnyaya L.V.* Paradigmy programmirovaniya. CH. 5. Uchebnyye yazyki programmirovaniya. Novosibirsk, 2015. 60 s. (Prepr. / ISI SO RAN; № 176).
URL: http://www.iis.nsk.su/files/preprints/gorodnyaya_176.pdf.

13. *Zvenigorodskiy G.A.* Pervyye uroki programmirovaniya. Bibliotechka «Kvanta». M.: Nauka, 1985. T. 41. URL: http://cip.iis.nsk.su/files/course/zven-ves.pdf.

14. *Kushnirenko A.G., Gribanova I.N., Rayko M.V., Zaydel'man Ya.N.* Kak my provodim vvodnoye zanyatiye po algoritmike v raznovozrastnoy gruppe doshkol'nikov i mladshikh shkol'nikov. URL:
https://fgoskomplekt.ru/upload/iblock/a2c/4o0ce5xyjyrha1b8qwsgxlhnjqp1iqme.pdf.

15. Chto takoye Logo. URL: http://www.int-edu.ru/logo/logo.html.

16. *Golikov D.V., Golikov A.D.* Kniga yunykh programmistov na Scratch – Smash-Words, 2013. ISBN 978-1310227554.

17. *Yershov A.P.* Programmirovaniye – vtoraya gramotnost'.
URL: http://ershov.iis.nsk.su/ru/second_literacy/article.

18. *Bruks F.* Mificheskiy cheloveko-mesyats, ili Kak sozdayutsya programmnyye sistemy. SPb.: Piter, 2021. 368 s.

19. *Gorodnyaya L.V.* O funktsional'nom programmirovanii // Komp'yuternyye instrumenty v obrazovanii (v pechati).

20. *Gorodnyaya L.V.* O probleme avtomatizatsii parallel'nogo programmirovaniya // Nauchnyy servis v seti Internet: mnogoobraziye superkomp'yuternykh mirov: Trudy Mezhdunarodnoy superkomp'yuternoy konferentsii (22–27 sentyabrya 2014 g., g. Novorossiysk). M.: Izd-vo MGU, 2014. S. 191–196.
URL: http://agora.guru.ru/abrau2014.

21. *Gorodnyaya L.V.* Abstraktnaya mashina yazyka programmirovaniya

uchebnogo naznacheniya SINKHRO // Vestnik Novosibirskogo gosudarstvennogo universiteta. Seriya: Informatsionnyye tekhnologii. 2021. № 4. S. 16–35.

22. *Adamovich A.I., Klimov And.V.* Ob opyte ispol'zovaniya sredy metaprogrammirovaniya Eclipse/TMF dlya konstruirovaniya spetsializirovannykh yazykov // Nauchnyy servis v seti Internet: trudy XVIII Vserossiyskoy nauchnoy konferentsii (19–24 sentyabrya 2016 g. g. Novorossiysk). M.: IPM im. M.V. Keldysha, 2016. S. 3–8. URL: https://doi.org/10.20948/abrau-2016-45.

23. *Gorodnyaya L.V.* Podkhody k predstavleniyu sintaksisa yazykov programmirovaniya. Novosibirsk, 2019. 39 s. (Prepr. / ISI SO RAN; № 185).
URL: https://www.iis.nsk.su/files/preprints/Preprint_185.pdf.

24. About SETL and GNU SETL. URL: https://setl.org/doc/setl.html.

25. *Schwartz Jacob T*. Set Theory as a Language for Program Specification and Programming. Courant Institute of Mathematical Sciences, New York University, 1970.

26. *Kondratyev D.A.*, *Promsky A.V.* Developing a self-applicable verification system. Theory and practice // Automatic Control and Computer Sciences. 2015. Vol. 49. Issue 7. P. 445–452. URL: https://doi.org/10.3103/S0146411615070123.

27. *Malyshkin V.E.* Tekhnologiya fragmentirovannogo programmirovaniya // Parallel'nyye vychislitel'nyye tekhnologii (PaVT'2012): trudy mezhd.unarodnoy nauchnoy konferentsii (Novosibirsk, 26–30 marta 2012 g.). Chelyabinsk: Izdatel'skiy tsentr YUUrGU, 2012. S. 598–605.
URL: http://omega.sp.susu.ru/books/conference/PaVT2012/short/212.pdf.

28. *Gorodskaya L.V.* Modeli raboty s pamyat'yu v uchebnom yazyke programmirovaniya SINKHRO // Nauchnyy servis v seti Internet: trudy XXIV Vserossiyskoy nauchnoy konferentsii (19–22 sentyabrya 2022 g., onlayn). M.: IPM im. M.V. Keldysha, 2022. S. 137–154.

29. *Gorodskaya L.V.* Rabota s dannymi v uchebnom yazyke programmirovaniya SINKHRO // Superkomp'yuternyye dni v Rossii: Trudy mezhdunarodnoy konferentsii. 26–27 sentyabrya 2022 g., Moskva / Pod. red. Vl. V. Voyevodina. M.: MAKS Press, 2022. ISBN 978-5-317-06875-2 e-ISBN 978-5-317-06876-9
URL: https://doi.org/10.29003/m3109.RussianSCDays2022. S. 87–97.
URL: https://doi.org/10.29003/m3109.RussianSCDays2022(vneshnyaya ssylka).

30. *McCarthy J.* LISP 1.5 Programming Mannual. The MIT Press., Cambridge, 1963. 106 p.

31. *Khenderson P.* Funktsional'noye programmirovaniye. M.: Mir, 1983. 349 s.

32. *Gorodnyaya L.V.* Gumanitarnyye faktory programmirovaniya / Sib. Otdeleniye Ros. Akad. Nauk, In-t sistem informatiki im. A.P. Yershova. Novosibirsk: Izd-vo SO RAN, 2020. 163 s.

33. *Gorodnyaya L.V.* Funktsional'noye programmirovaniye. Paradigma, modeli, metody / Sib. Otdeleniye Ros. Akad. nauk, In-t sistem informatiki im. A.P. Yershova. Novosibirsk: Izd-vo SO RAN, 2022. 482 s.

СВЕДЕНИЯ ОБ АВТОРЕ

ГОРОДНЯЯ Лидия Васильевна *– старший научный сотрудник Института систем информатики имени акад. Андрея Петровича Ершова СО РАН, доцент Новосибирского государственного университета, специалист в области системного программирования и образовательной информатики.*

Lidia Vasiljevna GORODNYAYA – Senior Researcher of A.P. Ershov Institute of Informatics Systems, Siberian Branch of the Russian Academy of Sciences, Associate Professor of Novosibirsk State University, a specialist in system programming and educational informatics.

email: lgorod@iis.nsk.su

ORCID: 0000-0002-4639-9032

Материал поступил в редакцию 10 декабря 2022 года