УДК 004.43 (042.4)

КАКИМ ДОЛЖЕН БЫТЬ ЯЗЫК УЧЕБНОГО ПРОГРАММИРОВАНИЯ

Л.В. Городняя[0000-0002-4639-9032]

Институт систем информатики им. А.П. Ершова СО РАН;
Новосибирский государственный университет;

[1]lidvas@gmail.com

Аннотация

Статья посвящена обоснованию решений в проекте тренажёра на базе языка учебного программирования, предназначенного для начального ознакомления с базовыми понятиями взаимодействия процессов и управления вычислениями. На этапе перехода к многопроцессорным архитектурам возрастает актуальность развития особой языково-информационной поддержки введения в программирование. Сколь ни сложен мир параллелизма, системе подготовки программистов предстоит его освоить и создать методику полноценного ознакомления с его не очевидными явлениями. Это достаточная причина для разработки языка учебного программирования, ориентированного на начальное обучение школьников младших и средних классов, а также студентов младших курсов и непрофессионалов, оперированию взаимодействующими процессами и программированию параллельных вычислений. В основу языка положен многолетний опыт управления взаимодействием игрушечных роботов, перемещающихся на клетчатой доске.

Материал статьи представляет интерес для программистов, студентов и аспирантов, специализирующихся в области системного и теоретического программирования, и для всех тех, кто интересуется проблемами современной информатики, программирования и информационных технологий, особенно проблемами параллельных вычислений, суперкомпьютерами и вообще применением многопроцессорных комплексов и компьютерных сетей.

Ключевые слова*: учебное программирование,* функциональное программирование*, взаимодействие процессов, многопроцессорные конфигурации,*

определение языка программирования, парадигмы программирования, декомпозиция программ, критерии декомпозиции, семантические системы, схемы преподавания системного программирования, лаконичные определения.

ВВЕДЕНИЕ

Появление и повсеместное распространение многопроцессорных и сетевых комплексов переводят информатику и программирование в мир параллелизма. Алгоритмы становятся параллельными, программы – многопоточными, компьютеры – многопроцессорными. Очередное действие может начать выполняться до завершения предыдущего, средства управления кроме логических значений используют события и многое другое [1]. И всё это происходит в многоязыковой обстановке – мощность пространства языков программирования уже перевалила за десятки тысяч[1]. Ещё в начале 1960-х при создании языка APL его автор, Кеннет Айверсон (*Kenneth Iverson)*, утверждал, что истинное программирование – это организация параллельных процессов [2]. Многие идеи организации процессов на уровне операционных систем слишком ограничены механизмами параллельной обработки на базе очередей и векторов [3]. Многочисленные прецеденты решения этой проблемы в рамках традиционных языков высокого уровня (ЯВУ) пока не привели к формированию парадигмы параллельного программирования, достаточной для полного решения этой проблемы [4]. Переход к многопроцессорным архитектурам требует дальнейшего развития языково-информационной поддержки параллельного программирования, особенно при введении в программирование. Можно обратить внимание, что в отличие от последовательного программирования организация параллельных вычислений (ПВ) не может полностью абстрагироваться от ряда низкоуровневых понятий, отражающих разнообразие ситуаций, возникающих при взаимодействии процессов, ознакомление с которыми естественно при обучении программированию [5–9]. Системе подготовки программистов предстоит освоить мир параллелизма и создать методику полноценного ознакомления с его не очевидными явлениями, даже если это не легко.

Целью статьи является не только наметить требования к современному языку учебного программирования, но и уяснить причины намеченных решений.

[1] https://www.levenez.com/lang/ Computer Languages History

Статья начинается с анализа тенденций в поисках средств и методов параллельного программирования, затем рассмотрены общие требования к реализации тренажёра для ознакомления с явлениями параллелизма, далее описаны особенности учебных многопоточных программ и в заключении отмечено текущее состояние проекта тренажёра на базе языка учебного программирования СИНХРОН, представляющего собой попытку согласованной реализации выработанных требований, ориентированного на ознакомление с явлениями параллелизма в форме разработки учащимися своих игр, выглядящих как взаимодействие роботов.

ОБЩИЕ ТЕНДЕНЦИИ

История языков программирования накопила ряд работоспособных идей по эффективному представлению естественного параллелизма при серийной обработке сложных структур данных (APL, Algol-68, SETL, Ada, БАРС, Sisal, Норма, mpC и др.). Для большинства таких ЯВУ характерно включение в семантику языка конкретной модели параллелизма. Так, функциональный язык APL поддерживает обобщение скалярных операций до обработки однородных векторов произвольной размерности [2]. Algol-68 предлагает управление процессами в терминах семафоров и критических участков, используя привычки последовательного программирования. SETL сводит параллелизм к математической независимости элементов множества [10]. Ada использует механизм «рандеву», требующий для обмена данными одновременное существование процессов. БАРС представляет взаимодействия процессов с помощью синхросетей над комплексами данных и выражений и поддерживает программирование разных дисциплин работы с памятью. Функциональный язык параллельного программирования Sisal предоставляет ряд средств формирования пространств итераций для параллельного исполнения циклов и для свёртки параллельно полученных значений в общий результат [11]. Язык Норма сосредоточен на проблеме представления сеточных вычислительных алгоритмов на основе непроцедурного описания решаемой задачи [12]. Интересен уникальный язык mpC, нацеленный на сетевое представление многопроцессорных конфигураций при выполнении многопоточных программ, допускающий синхронизацию потоков с помощью барьеров и перераспределение

нагрузки процессоров [13]. Производственные инструменты MPI и Open MP связаны с аппаратными моделями параллелизма, что чревато трудоемкостью известной проблемы мобильности программ. В таблице 1 показана хронологии некоторых решений по представлению параллельных вычислений.

Таблица 1.

Представление ПВ в некоторых ЯП

Год	ЯП	*Средства представления параллельных вычислений*
1960	**APL**	Приоритет матрицам, их деструктуризация и просачивание скалярных операций до обработки однородных векторов произвольной размерности.
1968	**Algol-68**	Управление процессами в терминах семафоров и критических участков, позволяющих параллелизм сводить к однопроцессорным программам.
1970	**SETL**	Независимость элементов множества и поддержка полного спектра теоретико-множественных методов работы с информацией.
1979	**Ada**	Синхронизация процессов, обменивающихся данными – «рандеву».
1984	**Clisp**	Мультизначения и пакеты – пространства имён.
1985	**БАРС**	Комплексы (размеченные множества с кратностью вхождения элементов), сети Петри и синхросети над комплексами данных и выражений, включая процессоры, просачивание, программируемые дисциплины доступа к памяти.
1986	**Sisal**	Базируется на языке Pascal, иерархия SSA-форм (участки однократного присваивания), матрицы, потоки, частичные функции, множественные результаты выражений, пространства итераций для параллельного исполнения циклов, просачивание, комплекты результатов выражений, свёртка, диагностические значения для всех типов данных .

1991	**Норма**	Сеточные вычислительные алгоритмы с выделением разных схем параллелизма.
1994	**MPI**	Обмен данными между процессами.
1995	**MpC**	Расширение C, многопроцессорные конфигурации, допускающие синхронизацию в терминах сетей и барьеров, поддерживает перераспределение нагрузки процессоров.
1997	**OpenMP**	Распараллеливание программ.
2007	**Clojure**	Указательные переменные, разные дисциплины доступа к памяти и стратегии параллелизма, деструктуризация списка параметров функции, синхронизация потоков, события, агенты.

В каждом языке поддержаны конкретные средства без интеграции со средствами из других языков. Тем не менее, при решении одной задачи могут реально понадобиться решения, доступные в других языках. Формально ЯВУ обладают полнотой по Тьюрингу, что означает принципиальную возможность реализации любых алгоритмов, включая параллельные.

Вопрос в трудоёмкости и эффективности нужных решений. Сравнение диалектов языка Lisp (Scheme, Clisp, Clojure) показывает, что в таких случаях происходит сначала моделирование недостающих средств на уровне семантики основного языка, затем реализуется их неявная поддержка на уровне прагматики в системах программирования для ряда диалектов и, наконец, достигается их явное включение в определение диалекта, обобщающего накопленный опыт. Такая динамика показана в Таблице 2 на примере механизма работы со списками свойств атома.

Примерно так в современные ЯП (Common Lisp, Haskell, F#, Kotlin, C#, Java, JavaScript, Scala, Python, Clojure) постепенно проникают понятия, отсутствующие или непонятые на уровне концепций более ранних языков, но проявляющиеся в работе отдельных функций (символы, откладывание, ожидание, обещание, задания, потоки, очереди, мьютексы, семафоры, атомы и др.). Подключаются функции высшего порядка и специальные функции, поддерживающие разметку участков

для распараллеливания и безопасного обмена данными. Появляются специальные примитивы, реакции на события, обратные вызовы и пересечение.

Таблица 2.

Развитие понятий в новых диалектах на примере списков свойств атомов.

Год	ЯП	*Список свойств атома – встроенная база данных*	*Особенности обработки списков свойств атомов*
1960	Lisp	(put atom flag value) ; *объявить значение* *; заданного свойства* (get atom flag) ; *найти значение* *; данного свойства*	– Функции PUT и GET образуют полную алгебраическую систему. – Объявлять свойство можно многократно, работает самое новое, сохраняя доступ к старому при необходимости. – При отладке можно манипулировать прежними определениями, подобно транзакционной памяти в базах данных.
1975	Scheme	(define (put atom property value)) (hash-set! properties (list atom property) value)) ; зада*ть значение* *; свойства атома* (define (get atom property)) (hash-ref properties (list atom property) #f)) ; взять *значение свойства*	Для программируемых свойств можно использовать хэш-таблицы, приспособленные к хранению пар «ключ-значение», чтобы функциями hash-set! и hash-ref получать эффект списков свойств. Встроенные средства работают только со значениями переменных и определениями функций.
1984	Common Lisp	(setf (get atom flag) value) *; объявить значение* *; данного свойства (put)* (get atom flag)	– Используется или порождается **адресуемое поле**, значение которого можно безвозвратно изменять (setf).

		; найти значение *; заданного свойства*	– На уровне языка отсутствуют понятия «поле» и «адрес», они неявно возникают при исполнении программы. – Повышается эффективность отлаженных программ.
2007	Clojure	(def newatom *; объявить атом* (atom {:name "Hydrogen" :atomic-number 1})) (swap! newatom assoc :property "value") ; *дать значение свойства* @newatom *; значение свойств атома* *; – как разыменование* (swap! newatom dissoc :property) *; удалить свойство* (reset! newatom {:property "value"}) *; изменить свойство*	Атом реализован как указатель на список его свойств. В этот список можно функцией swap! размещать новые свойства (assoc) или их удалять (dissoc), а функцией reset! изменять. Старое свойство в таком случае становится недоступным, но не исчезает, можно организовать его восстановление, сохранив заранее при необходимости, что близко механизму транзакционности.

Встречается совмещение возможностей, фильтры, отображения, изменение значений, а также методы и интерфейсы, контроль состояний, дополнительные парадигмы, реактивное программирование, побочные эффекты и асинхронность. На уровне системной поддержки предоставляется манипулирование временем, асинхронное выполнение, фоновый режим и блокировки вычислений, разделение пространств имен, порождение специализированных версий, включение иноязычных модулей и специализированных систем с управлением обработкой ошибок и выбором методов, включая параллелизм через создание объектов.

Как и при моделировании разных парадигм, привычные понятия при переходе к ПВ влекут необходимость в их расширении или во введении дополнительных понятий, а также во вспомогательных синтаксических формах, включаемых в определение ЯВУ для представления программ организации параллельных процессов, обладающих более длительным жизненным циклом, чем последовательные программы, и работающих в более широком пространстве, осознание которого в учебном процессе легче, чем в производственном.

ВЫБОР ОСНОВНЫХ РЕШЕНИЙ

Проблемой является создание в рамках единой среды и языковой модели перспективы пошагового движения к полному своду решений, имеющих прецеденты в разных языках и системах программирования. Выбор любой одной модели параллельных вычислений при обучении может оказаться неудачным из-за стремительного развития элементной базы и ИТ, особенности которого не всегда удаётся предвидеть. Поэтому полезно предусматривать развитие или уточнение учебных моделей и основных понятий программирования, выделяя фрагменты выражений, действий, вычислений и данных, включая процессоры. Проект реализации тренажёра по учебному программированию СИНХРОН отражает первые шаги процесса обучения, начинающегося оперированием многопоточной программой с приоритетом параллелизму и затем шагами многопоточного и синхронизирующего программирования с элементами метапрограммирования.

Оперирование программой необходимо для ознакомления с наиболее трудными проблемами синхронизации процессов. По умолчанию тренажёр содержит интерпретатор, обеспечивающий оперирование выполнением программ, сопровождаемое протоколом успешных действий. При выполнении заданий одновременно с решением учебной задачи в форме взаимодействия потоков должен быть построен контекст, в котором синхронизация потоков корректна, иначе программа выполняется до обнаружения невыполнимого действия и приостанавливается. Подобный подход опробован в языках и системах Робик, Logo, Karel, Кумир, Роботландия, Вертун, ПиктоМир и других. Возможно переключение в режим редактирования программы с использованием протокола, подобно системам Delpfi, Е-практикум, Рапира.

Приоритет параллелизму позволяет опередить привычку к последовательному программированию**.** Параллельные вычисления показали себя трудной задачей, требующей особой парадигмы, а новые парадигмы требуют предварительного иллюстративного материала для активизации и целенаправленного формирования скрытых моделей изучаемой парадигмы, отвечающей на вызовы новых проблем. Прецеденты в языках APL, SETL, БАРС, Sisal, mpC, Lisp2D[2] и других [1, 3, 4].

Мультипарадигмальность исключает чрезмерное привыкание к одной парадигме. Языки XXI века, особенно долгоживущие и учебные, как правило, поддерживают все основные и фундаментальные парадигмы программирования, что позволяет программировать решения задач разной степени изученности, с разной длительностью жизненного цикла постановки задачи в рамках единой обстановки и получать навыки работы на всех фазах полного жизненного цикла программ, как в языках Lisp и Ada. Стихийно сформированные интуитивные модели и сложившиеся в обычной жизни и учёбе навыки успешны при обучении императивно-процедурному и объектно-ориентированному программированию при решении типовых задач. Они не достаточны при обучении функциональному, параллельному и логическому программированию, а также новым парадигмам, возникающим при обнаружении новых, особо сложных, трудно решаемых задач, подход к которым может показать опыт применения тренажёра [5, 6].

Функциональное программирование в силу своей универсальности приспособлено к моделированию разных парадигм программирования, что показано в Таблице 3 на диалектах языка Lisp.

Таблица 3.

Пример функционального моделирования парадигм программирования

Парадигма	*Базовые средства и понятия*
Чистое функциональное программирование (Pure Lisp)	Программа строится из представлений обычных и специальных функций.

[2] http://lisp2d.net/rus/ppfs.html Глебов А.Н. Параллельное программирование в функциональном стиле

	Поддерживается динамическая область видимости локальных переменных в соответствии с иерархией вызовов функций (ассоциативный список). Выбор ветви при ветвлении требует наличия истинного предиката. Ядро языка содержит обычные (Cons, Car, Cdr, Eq, Atom, Eval) и специальные (Lambda, Label[3], Cond, Quote) функции над списками, представляющими любые конструкции языка. Для компиляции определена абстрактная машина SECD, регистры которой приспособлены для хранения результатов (S), значений переменных и определений функций (E), программы управления вычислениями (C) и защиты восстанавливаемых данных от случайных изменений (D).
Императивно-процедурное программирование (Lisp 1.5)	Введены специальные функции Prog, Setq, Return, Go – аналоги привычных императивных конструкций. Смягчено ветвление – результат ветвления Cond, определен и без истинного предиката. Появляются независимые области видимости для рабочих переменных и передач управления по меткам/тэгам. Поддержаны упрощенные варианты вычисления линейных участков выражений Prog1, Prog2, ProgN.
Логическое программирование (Недетерминизм)	Используется понятие «вариант» как структура, формально не образующая иерархии. Добавлено специальное логическое значение «***ESC»*** (*ТУПИК*) – его действие заключается в том,

[3] Define или Defun в производственных диалектах.

	что оно как бы «старается» по возможности не вычисляться, символизируя неподходящий вариант и переход к бэктрекингу — перебор вариантов. Логика «истина–ложь» подменяется логикой «успех–отказ», характерной для операционных систем, в которых ненулевое значение кода завершения действий используется для доступа к обработчикам прерываний, обеспечивающим продолжение функционирования системы, подобно функции error в Lisp 1.5 или Racket. Для сокращения перебора вариантов логика «успех–отказ» даёт жизнь методам сопоставления с образцами и защитным условиям. В определение абстрактной машины SECD добавлен регистр R для хранения независимых вариантов, получается машина SECDR. В систему команд абстрактной машины добавляются команды выбора любого варианта и реагирования на неподходящий вариант (any, esc), что можно применять и для реагирования на неожиданности (ловушки и обработка прерываний).
Ленивые вычисления: отложенные или опережающие (Pure Lisp)	Определена конструкция «рецепт» как замыкание функции, поддерживающая оптимизацию – исключение дублирования вычислений. Определены функции приостановки и возобновления вычислений, эквивалентные созданию функции без параметров и вычислению тела функции из её замыкания. В систему команд абстрактной машины SECD добавляются команды для эффективной работы с рецептами (LDE, RPL, APN).

	Полученная техника позволяет работать с бесконечными структурами данных, некоторыми неопределенностями, проекциями и частичными вычислениями, а также организовывать мемоизацию.
Многопоточность (Common Lisp)	Вводится понятие «мультизначение» как расширение понятия «значение» допущением дополнительных значений, независимо формируемых операциями или функциями. Определены специальные функции перехода от представленной мультизначениями модели комплекта потоков в обычное значение и обратного преобразования структуры данных в комплект потоков. Поддерживается связывание элементов мультизначения с переменными, применение к ним функций и другие методы взаимодействия программы с потоками. Порядок выполнения потоков не определён, что поддерживается механизмом, похожим на упрощённую работу вариантов без ESC и бэктрекинга.
Метапрограммирование (Common Lisp)	Механизм определения специальных функций конкретизируется как макротехника над списками, позволяющая формировать определения функций, учитывающие особенности условий их применения, что эквивалентно конструированию функцией Cons символьного представления определения функции из атомов, входящих в это определение.
ООП (Common Lisp - CLOS)	Введены специальные функции, аналогичные базовым средствам работы с классами объектов (defclass, make-instance, slot-value, defmethod, defgeneric), использующие иерархию хэш-таблиц,

подобных ассоциативным спискам, или списки свойств атомов как механизм наследования.

Механизм перегрузки методов (полиморфизм) сведён к варьированию числа и категорий параметров функций, выделяя позиционные, ключевые, необязательные и серийные параметры.

Роль инкапсуляции выполняет механизм замыкания функций, приспособленный к совместному хранению функций и значений свободных переменных.

Можно обратить внимание, что решения на приаппаратном уровне обусловлены соображениями эффективности, пресс которых для небольших учебных программ не является решающим. Следовательно, для учебного языка достаточно поддержать парадигму функционального программирования и в тематику лабораторных работ включить моделирование всех необходимых парадигм.

Функциональные прототипы параллельных программ полезны в качестве учебных образцов для лабораторных работ после опыта оперирования роботами на клетчатой доске. Языки параллельного программирования занимают видное место среди функциональных языков. Программы на таких языках обычно свободны от побочных эффектов и ошибок, непредсказуемо зависящих от реального времени. Это существенно упрощает отладку программ ПВ благодаря выделению автономных фрагментов подобно мультизначениям или пакетам в Clisp [7–9].

Ленивые и опережающие вычисления используются как средство оптимизации параллельных программ**.** Результативность вычислений можно повышать с помощью специальноустроенных данных – так называемых «рецептов» и мемоизации. Рецепт предназначен для отложенного или опережающего исполнения фрагмента программы (ленивые вычисления) и представлен как конструкция из функции и контекста для её выполнения, называемая замыканием функции. Мемоизация позволяет исключать дублирование сложных вычислений сохранением ранее полученных результатов. Такая техника доступна в языках Lisp, Setl, Clisp. Выполнение параллельных процессов часто связано с независимостью порядка вычислений от последовательности представления действий в программе [7, 10].

Расширяемость языков и систем программирования при экспериментах по созданию учебных игр с оригинальным интерфейсом позволяет оценить возможности улучшать программы. При реализации учебного языка ПВ, могут пригодиться идеи языков Lisp, F#, C#, допускающих динамическую обработку кода программ в рамках парадигмы функционального программирования. Это позволяет обеспечить лаконизм представления программ, облегчить отладку программ, использование инородных модулей и упростить реализацию специальных версий интерпретатора, компилятора и виртуальной машины языка. Решения проблемы расширяемости или настраиваемости ЯВУ становятся актуальнее по мере развития средств и методов ПВ, обусловленного высоким темпом прогресса в области элементной базы и информационных технологий, они различаются в системах с препроцессорами или языках Lisp и Setl [4].

Редактирование фрагментов, возможно обладающих необычной раскладкой понятий, не соответствующих используемому языку, полезно для понимания, что существуют малозаметные различия на разных уровнях и в разных языках. Кроме простого сцепления текстов при подготовке и отладке программ используются синтаксические макросы, вид параметров которых следует задавать с помощью синтаксического подобия вхождению переменной в заданную строку согласно синтаксису языка. Похожий контроль можно налаживать в языке JavaScript с помощью функции EVAL, вычисляющей выражения на уровне исходного текста. Это можно рассматривать как задачу выбора форм или шаблонов проектирования для представления и организации семейств допустимых параллельных процессов на примере языков Setl и Clisp [14].

Фрагментация программ и синтаксическое подобие возникают при декомпозиции программы ПВ на схему управления и её наполнение. При определении функций кроме обычных переменных выделяются так называемые «фрагментные», используемые для параметризации схем или шаблонов проектирования и наполнения их подходящим содержанием, возможно требующим контроля вида фрагмента. Все вхождения фрагментной переменной в схему должны быть синтаксически эквивалентны её вхождению в строку, задающую вид значения переменной при определении макроса в отличие от типа переменной при определении функции [14, 15].

Трансформационная семантика, задающая семейства функционально эквивалентных процессов, позволяет уяснять варианты допустимых решений. Общие решения по обеспечению лаконизма, универсальности, конструктивности и расширяемости приводят к трансформационной семантике ЯВУ, определяющей возможные преобразования программ, что является естественным полигоном для метапрограмирования, поддержанного в языках Lisp и его диалекте Clisp [16].

Преобразование программы позволяют частично изменять её свойств при сохранении отлаженных фрагментов или исключении нежелательных оптимизаций. Часть сложностей обучения практическому параллельному программированию вытекает из достаточно очевидного сдвига понятий, вызванного тем, что реально не существует возможности использовать ЯВУ в чистом виде. Системы программирования подвергают программы разным, не вполне эквивалентным, оптимизирующим преобразованиям, увидеть и осознать эффекты которых полезно в учебной практике. Такие примеры для многопоточных программ могут быть связаны с распределением действий по процессорам или с синхронизацией потоков действий, что технически выполнимо на базе Lisp, Clisp и других языков функционального программирования [16].

Метапрограммирование для преобразования программ полезно при их отладке и прототипировании. Достаточно простые преобразования сети потоков позволяют варьировать схемы выполнения вычислений и сводить многие конструкции языка программирования к взаимодействию простых потоков. Обратимость преобразований и чувствительность их результата к информационным связям между фрагментами программы требуют умения формализовать критерии применимости трансформаций и выбора подходящего варианта, включая перераспределение нагрузки как в mpC. Метапрограммирование при решении таких задач в основном может использовать символьную обработку, дополненную средствами проверки синтаксической эквивалентности и верификации программ [15, 16].

СПЕЦИФИКА МНОГОПОТОЧНОСТИ

Проект языка СИНХРОН поддерживает переход к многопоточности и более сложному параллелизму, резко расширяя пространство допустимых процессов

выполнения программ и разнообразие архитектур. Возникают вопросы взаимодействия локальной и общей памяти и фильтрации данных при обработке структур данных, особенно больших данных. Сложность вычислений может быть замаскирована просачиванием операций по структурам данных. Отладка взаимодействующих процессов происходит в условиях варьирования порядка асинхронных действий и отлаженных или верифицированных фрагментов. Управление рассредоточенными действиями приводит к учёту событий и условий готовности процессоров. Дополнительные сложности вытекают из задач синхронизации фрагментов программы. Всё это зависит от навыков предвидеть проблемы, выходящие за пределы первичного представления программ и их прототипов.

Отделение порядка вычислений от порядка представления выражений позволяет учитывать пространство возможных процессов, заметно расширяющееся при переходе к многопоточным программам. Для развития навыков сознательного выбора решений, влияющих на изменение данных в общей памяти, в представлении структур данных учебного языка СИНХРОН предложено разнести смысл скобок и разделителей. Скобки означают порядок доступа к элементам структур данных в памяти, а разделители – порядок вычисления элементов при исполнении программы. В результате структуры данных наполняются элементами, порядок вычисления которых может отличаться от порядка вхождения в программу. Многопоточная программа допускает асинхронное выполнение потоков и действий, что означает независимость порядка вычисления элементов выражений от порядка их вхождения в структуры данных, рассматриваемые как функции над этими выражениями, что выполнимо в рамках модели языков Lisp и Prolog.

Просачивание операций и функций позволяет строить компактные представления программ. Основные методы лаконичного представления массовых вычислений связаны с использованием неявных циклов, позволяющих избежать выписывания однотипных схем над стандартными структурами данных типамногомерных векторов. Так, например, результат операции над скалярамив языках параллельного программирования обычно автоматически распространен на произвольные однородные структуры данных, что поддержано в языках APL, БАРС,

Sisal. Этот механизм в языке СИНХРОН естественно распространяется на технику применения функций, что повышает лаконизм выражений [2, 11, 15].

Фильтрация данных упрощает и повышает эффективность определения обработчиков больших данных. Из бинарных операций можно конструировать фильтры. Результат фильтрации исчезает из аргумента – он переносится в другую структуру данных, подобно обработке множеств в языке теоретико-множественного программирования SETL [10], что удобно при подготовке программ алгоритмов перебора. Структура из фильтров создает структуру из результатов их применения к одному и тому же аргументу [15].

Разнообразие архитектур, изученное в процессе ознакомления с параллелизмом, способствует пониманию механизмов, влияющих на эффективность ПВ. Проблемы параллельного программирования дополнительно осложнены дистанцией между уровнем абстрактных понятий, в которых описываются решения сложных задач, и уровнем конкретных аппаратных средств и архитектурных принципов управления параллельными вычислениями (потактовая синхронизация, совмещение действий во VLIW-архитектурах, сигналы, семафоры, буферы со временем ожидания сообщения, прерывания ит. п.). Проблемы подготовки параллельных программ для всех столь разных моделей обладают общностью, но есть и существенная специфика, требующая понимания разницы в критериях оценки программ и информационных систем для различных применений, что может быть показано на программируемых моделях [1, 4, 17].

Взаимодействующие процессы – естественный полигон оптимизации ПВ**.** Популярная модель Т. Хоара чувствительна к обнаружению достаточно тонких ошибок при создании программ взаимодействия процессов, её нередко используют в системах верификации свойств программ [3]. Разработка таких программ отличается от подготовки обычных программ на весьма глубоком уровне, что можно показать при оперировании роботами на специальном диалекте языка СИНХРОН [15, 18, 19].

Варьирование контекста асинхронных действий даёт навыки обеспечения универсальности фрагментов программ. Текст программы на языке ПВ строится в определённой обстановке или контексте из действий – выражений или директив,

использующих данные из контекста, выполняющего роль общей памяти. Выражения могут использовать размещённые в общей памяти данные, но не изменяют их. Директивы могут изменять хранимые в общей памяти данные, сохраняя доступ к устаревшим данным, как это поддержано неявным механизмом транзакционной памяти в языке Clojure, использующим принцип неизменяемости данных в языках функционального программирования.

Взаимодействие локальной и общей памяти ждёт практичных решений, открывающих программирование процессов, использующих побочные эффекты ради эффективности. Работа с данными при организации ПВ потребовала дополнительных решений. На уровне ядра языка необходим механизм, отчасти смягчающий проблемы освобождения памяти для многопоточных программ [15, 18, 19]. Обычная система команд виртуальной машины для языка СИНХРОН пополнена средствами для решения проблем работы с общей памятью и внешними устройствами. Это команды пересылок данных и обмена данными в общей памяти многопроцессорного комплекса, нужные для профилактики возникновения временных интервалов между взаимосвязанными присваиваниями в общей памяти. Возможен побочный эффект присваивания, допускающий при необходимости восстановление прежних значений переменных – транзакционная память или персистентность, характерная для операционных систем и баз данных. Кроме того, предложена реализация императивной синхронизации взаимодействия локальной и общей памяти как пары запрос и его двойник, выполняемые неразрывно в стиле рандеву языка Ada.

События и условия готовности выполняют роль синхронизаторов сети процессов. При выполнении многопоточной программы достижение некоторых позиций рассматривается как событие. Объявление события позволяет выполняться ждущим его действиям, обладающим готовностью. Событие не сбрасывается, пока все ждущие его потоки не будут готовы или их выполнение не будет оценено как невозможное. Независимые описания процессов можно связывать в терминах разметки барьерами – синхросети. Узлы с одинаковой разметкой срабатывают одновременно. Полное представление об асинхронных процессах, их эффективности и проблемах организации дают работы по сетям Петри в языке БАРС.

По мере выполнения действия в языке СИНХРОН формируются сообщения о готовности к выполнению, т. е. началу, о собственно выполнении и о завершении действия. 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.

Синхронизация слоёв программы позволяет программу рассматривать как конструкцию их независимых потоков, распределённых по частично упорядоченным слоям, точнее многопоточная программа представляется как набор потоков и может быть размечена барьерами на слои. Каждый поток состоит из очереди вычисляемых, возможно, помеченных барьерами, действий. Одноимённые барьеры выполняют роль точек синхронизации, разбивающих программу на слои. Каждый слойначинает выполняться одновременно и завершается до выполнения следующего слоя. Выражения одного слоя из разных, представленных независимо, потоков с одинаковой пометкой выполняются одновременно, как в синхросетях языка БАРС.

Навыки предвидеть проблемы, связанные со сложностью организации параллельных процессов, следует формировать в учебном процессе до включения в производственную деятельность. Часть проблем взаимодействия потоков алгоритмически не разрешима, поэтому в задачи ознакомления с параллелизмом входит научиться обнаруживать, предвидеть и отлаживать программы при обнаружении неудачно взаимодействующих потоков, что в своё время было проиллюстрировано в языке Робик. Наполнение многопоточной программы может развиваться независимо от схем управления вычислениями в отдельных потоках, а схемы можно реорганизовывать без дополнительной отладки наполнения в соответствии с принципом факторизации на автономно развиваемые модули. Схемы работают подобно макросам, но с контролем соответствия параметров объявленным видам фрагментов. Директива при благополучном исходе обработки памяти даёт результат подобно выражению.

Когнитивные ошибки, встречающиеся в любой деятельности, иногда бывают спровоцированы некоторыми конструкциями ЯВУ. Одной из проблем при создании игр детьми является трудность представления и понимания сложных

логических и вероятностных условий управления действиями персонажей. Эта проблема обусловлена противоречием между интуитивной и формальной оценкой истинности логических и вероятностных формул. Интуитивно истинность или вероятность связки «&&» оценивается выше, чем истинность составляющих, а реально и формально она ниже. Это противоречие, возможно имеющее лингвистический характер, замечено в исследованиях нобелевского лауреата Д. Канемана, утверждающего, что математическое образование от таких ошибок спасает редко [20]. В языке SETL предлагалось решение этой проблему для ветвлений с помощью таблиц решений, применяемых в бухгалтерии. Похожая техника в языке СИНХРОН реализуется с помощью специального двумерного табличного синтаксиса, похожего на стиль языка Python.

ЗАКЛЮЧЕНИЕ

Проект разработки тренажёра для обучения программированию на базе учебного языка СИНХРОН представляет собой эксперимент по выбору базовых средств для достаточно полного решения проблем обучения эффективной реализации параллельных алгоритмов, вынужденно требующих использовать весьма широкий спектр сложно совместимых средств: от управляющих действий более низкого уровня, чем в привычных ЯВУ, до манипулирования пространствами решений по обработке данных в памяти, типичных для языков сверх высокого уровня [10]. Обзор исследований и решений в смежных областях представлен в [3, 4].

Особый круг учебных 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 для конкретной архитектуры. Новые, учебные и долгоживущие языки программирования, как правило, имеют экспериментальный, динамический и мультипарадигмальный характер, что приводит к идее их расширения на основные модели параллельных вычислений и архитектур. Нужна система обучения, поддерживающая осознание особенностей взаимодействия процессов, удобная

для экспериментов по формированию навыков подготовки работоспособных программ и понимания новых возможностей аппаратуры.

К настоящему времени в рамках специализации студентов на базе ФИТ и ММФ НГУ выполнена реализация виртуальной машины на базе языка Clojure, достаточной для поддержки специфики многопоточных программ при ознакомлении с разными явлениями параллелизма, и визуальная оболочка на базе языка Kotlin, поддерживающая экспериментальную разработку учебных игр[4]. Предстоит реализация диалектов для асинхронного выполнения независимых потоков программ и синхронизации взаимодействующих потоков при подготовке учебных многопоточных программ.

СПИСОК ЛИТЕРАТУРЫ

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

2. *Магариу Н.А.* Язык программирования АПЛ. М.: «Радио и связь», 1983. 96 с.

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

4. *Марчук А.Г., Городняя Л.В.* Развитие моделей параллелизма в языках высокого уровня // Научный сервис в сети Интернет: все грани параллелизма: Труды Международной суперкомпьютерной конференции (23–28 сентября 2013 г., г. Новороссийск). М.: Изд-во МГУ, 2013. С. 342–346**.**
URL: http://agora.guru.ru/abrau2013/pdf/342.pdf

5. *Городняя Л.В.* От трудно решаемых проблем к парадигмам программирования // XXVI Байкальская Всероссийская конференция с международным участием «Информационные и математические технологии в науке и управлении» (июль 2021 года, Иркутск)**.** С**.** 94–109.
URL: https://cyberleninka.ru/article/n/ot-trudno-reshaemyh-problem-k-paradigmam-programmirovaniya

[4] Реализован уровень абстрактной машины языка СИНХРОН и оболочки студентами ФИТ и ММФ НГУ Д.В. Мажугой и М.Н. Дубковым.

6. *Городняя Л.В.* О неявной мультипарадигмальности параллельного программирования // Научный сервис в сети Интернет: труды XXIII Всероссийской научной конференции (20–23 сентября 2021 г.). М.: ИПМ им. М.В. Келдыша, 2021. С. 104–116**.** https://doi.org/10.20948/abrau-2021-6
URL: https://keldysh.ru/abrau/2021/theses/6.pdf

7. *Городняя Л.В.* О функциональном программировании // Журнал «Компьютерные инструменты в образовании» 2021, выпуск 3. С. 57–75**.**
URL: http://ipo.spb.ru/journal/index.php?article/2288/

8. *Городняя Л.В.* Место функционального программирования в организации параллельных вычислений // Информационные и математические технологии в науке и управлении. 2022. Выпуск №1(25). С. 102-119.
URL: https://www.imt-journal.ru/archive/public/article?id=230;
URL: https://cyberleninka.ru/article/n/mesto-funktsionalnogo-programmirovaniya-v-organizatsii-parallelnyh-vychisleniy https://doi.org/10.38028/Б81.2022.25.1.009

9. *Городняя Л.В.* Перспективы функционального программирования параллельных вычислений // Электронные библиотеки. 2021. Т. 24, № 6. С. 1090–1116.
URL: https://rdl-journal.ru/article/view/713

10. *Schwartz Jacob T.* Abstract algorithms and a set theoretic language for their expression // Computer Science Department, Courant Institute of Mathematical Sciences, New York University. 1971**.**
URL: https://www.softwarepreservation.org/projects/SETL/setl/doc/Schwartz-Abstract_Algorithms-1971.pdf

11. *Cann D.C.* SISAL 1.2: A Brief Introduction and tutorial // Preprint UCRL-MA-110620. Lawrence Livermore National Lab., Livermore, California, May, 1992. 128 p.

12. *Андрианов А.Н.* Сайт проекта Норма. https://keldysh.ru/pages/norma/

13. *Ластовецкий А.Л.* Программирование параллельных вычислений на неоднородных сетях компьютеров на языке mpC (Интерактивный учебный курс).
URL: https://parallel.ru/tech/mpc/mpC-rus.html

14. *Малышкин В.Э.* Технология фрагментированного программирования.
URL: http://omega.sp.susu.ru/books/conference/PaVT2012/short/212.pdf

15. *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. С. 87–97. https://doi.org/10.25205/1818-7900-2021-19-4-16-35

16. *Адамович И.А., Климов Ю.А.* Специализация интерпретаторов на объектно-ориентированных языках может быть эффективной // Научный сервис в сети Интернет: труды XXIV Всероссийской научной конференции (19–22 сентября 2022 г., онлайн). М.: ИПМ им. М.В. Келдыша, 2022. С. 3–24.
URL: https://keldysh.ru/abrau/2022/theses/18.pdf

17. *Андреева Т.А., Городняя Л.В.* Можно ли измерять вклад программистских решений в производительность программ? // Научный сервис в сети Интернет: труды XXV Всероссийской научной конференции (18–21 сентября 2023 г., онлайн). М.: ИПМ им. М.В. Келдыша, 2023. С. 12–24.

18. *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.

19. *Городняя Л.В.* Абстрактная машина языка программирования учебного назначения СИНХРО // Вестник НГУ. Серия: Информационные технологии**.** 2021, Т. 19, №4. С. 16–35*.* https://doi.org/10.25205/1818-7900-2021-19-4-16-35

20. *Канеман Д.* Думай медленно … решай быстро (Thinking, Fast and Slow). М.: АСТ, 2013*.* 625 с.

SELECTING SOLUTIONS IN AN EDUCATIONAL PROGRAMMING LANGUAGE SIMULATOR

L.V. Gorodnyaya[0000-0002-4639-9032]

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

lidvas@gmail.com

Abstract

The article is devoted to the development of solutions in the project of a simulator for teaching programming, intended for initial familiarization with the basic concepts of process interaction and calculation management. No matter how complex the world of parallelism is, the programmer training system will have to master it and create a methodology for fully familiarizing itself with its non-obvious phenomena. The simulator is based on the experience of controlling the interaction of toy robots moving on a checkered board. The article material is of interest to programmers, students and graduate students specializing in the field of system and theoretical programming.

Keywords*: educational programming, functional programming, process interaction, multiprocessor configurations, definition of a programming language, programming paradigms, program decomposition, decomposition criteria, semantic systems, schemes for teaching system programming, laconic definitions.*

REFERENCES

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

2. *Magariu N.A.* Yazyk programmirovaniya APL. M.: «Radio i svyaz'», 1983. 96 s.

3. *Hoare C.A.R.* Vzaimodeystvuyushchiye posledovatel'nyye protsessy. M.: Mir, 1989. 264 s.

4. *Marchuk A.G., Gorodnyaya L.V.* Razvitiye modeley parallelizma v yazykakh vysokogo urovnya // Nauchnyy servis v seti Internet: vse grani parallelizma: Trudy Mezhdunarodnoy superkomp'yuternoy konferentsii (23–28 sentyabrya 2013 g., g. Novorossiysk). M.: Izd-vo MGU, 2013. S. 342–346.

URL: http://agora.guru.ru/abrau2013/pdf/342.pdf

5. *Gorodnyaya L.V.* Ot trudno reshaemyh problem k paradigmam programmirovaniya // XXVI Bajkal'skaya Vserossijskaya konferenciya s mezhdunarodnym uchastiem «Informacionnye i matematicheskie tekhnologii v nauke i upravlenii» (iyul' 2021 goda, Irkutsk). S. 94–109.
URL: https://cyberleninka.ru/article/n/ot-trudno-reshaemyh-problem-k-paradigmam-programmirovaniya

6. *Gorodnyaya L.V.* O neyavnoy mul'tiparadigmal'nosti parallel'nogo programmirovaniya // Nauchnyy servis v seti Internet: trudy XXIII Vserossiyskoy nauchnoy konferentsii (20–23 sentyabrya 2021 g.). M.: IPM im. M.V. Keldysha, 2021. S. 104–116.
https://doi.org/10.20948/abrau-2021-6
URL: https://keldysh.ru/abrau/2021/theses/6.pdf

7. *Gorodnyaya L.V.* O funktsional'nom programmirovanii // Zhurnal «KIO» 2021, vypusk 3. S. 57–75. URL: http://ipo.spb.ru/journal/index.php?article/2288/;
URL: https://cyberleninka.ru/article/n/o-funktsionalnom-programmirovanii

8. *Gorodnyaya L.V.* Mesto funktsional'nogo programmirovaniya v organizatsii parallel'nykh vychisleniy // IMT v nauke i upravlenii. 2022. Vypusk №1(25). S. 102–119.
https://doi.org/10.38028/B81.2022.25.1.009,
URL: https://www.imt-journal.ru/archive/public/article;
URL: https://cyberleninka.ru/article/n/mesto-funktsionalnogo-programmirovaniya-v-organizatsii-parallelnyh-vychisleniy.

9. *Gorodnyaya L.V.* Perspectives of Functional Programming of Parallel Computations// Russian Digital Libraries Journal. 2021. V. 24. No.6. P. 1090–1116.
https://doi.org/10.26907/1562-5419-2021-24-6-1090-1116.
URL: https://rdl-journal.ru/article/view/713

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

11. *Cann D.C.* SISAL 1.2: A Brief Introduction and tutorial. Preprint UCRL-MA-110620. Lawrence Livermore National Lab., Livermore–California, May, 1992. 128 p.

12. *Andrianov A.N.* Sayt proyekta Norma. URL: https://keldysh.ru/pages/norma/

13. Lastovetskiy A.L. Programmirovaniye parallel'nykh vychisleniy na neodnorodnykh setyakh komp'yuterov na yazyke mpC (Interaktivnyy uchebnyy kurs).
URL: https://parallel.ru/tech/mpc/mpC-rus.html

14. *Malyshkin V.E.* Tekhnologiya fragmentirovannogo programmirovaniya.
URL: http://omega.sp.susu.ru/books/conference/PaVT2012/short/212.pdf

15. *Gorodnyaya 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. S. 87–97. https://doi.org/10.29003/m3109.RussianSCDays2022

16. *Adamovich I.A., Klimov Yu.A.* Spetsializatsiya interpretatorov na ob"yektno-oriyentirovannykh yazykakh mozhet byt' effektivnoy // Nauchnyy servis v seti Internet: trudy XXIV Vserossiyskoy nauchnoy konferentsii (19–22 sentyabrya 2022 g., onlayn. M.: IPM im. M.V. Keldysha, 2022. S. 3–24. https://doi.org/10.20948/abrau-2022-18,
URL: https://keldysh.ru/abrau/2022/theses/18.pdf

17. *Andreyeva T.A., Gorodnyaya L.V.* Mozhno li izmeryat' vklad programmistskikh resheniy v proizvoditel'nost' programm? // Nauchnyy servis v seti Internet: trudy XXV Vserossiyskoy nauchnoy konferentsii (18–21 sentyabrya 2023 g., onlayn). M.: IPM im. M.V. Keldysha, 2023. S. 12–24**.** https://doi.org/10.20948/abrau-2023-2
URL: https://keldysh.ru/abrau/2023/theses/2.pdf

18. *Gorodnyaya L.V.* Modeli raboty s pamyat'yu v uchebnom yazyke programmirovaniya SINKHRO // Nauchnyy servis v seti Internet: 2022. S. 137–154.

19. *Gorodnyaya L.V.* Abstraktnaya mashina yazyka programmirovaniya uchebnogo naznacheniya SINKHRO // Vestnik NGU. Seriya: Informatsionnyye tekhnologii. 2021 T. 19, №4. C. 16–35.
https://doi.org/10.25205/1818-7900-2021-19-4-16-35

20. *Kaneman D.* Dumay medlenno ... reshay bystro (Thinking, Fast and Slow). M.: AST, 2013. 625 s.

СВЕДЕНИЯ ОБ АВТОРЕ

ГОРОДНЯЯ Лидия Васильевна *– к. ф.-м. н., 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 at the A.P. Ershov Institute of Informatics Systems, Siberian Branch of the Russian Academy of Sciences, Associate Professor at the Novosibirsk State University, specialist in system programming and educational informatics.

e-mail: gorod@iis.nsk.su

ORCID: 0000-0002-4639-9032

Материал поступил в редакцию 27 октября 2024 года