Российская академия наук
Сибирское отделение
Институт систем информатики
им. А. П. Ершова

Л.В. Городняя

ЯЗЫК ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ СИНХРО, ПРЕДНАЗНАЧЕНЫЙ ДЛЯ ОБУЧЕНИЯ

Препринт
180

Новосибирск 2016

Препринт посвящен проекту языка обучения параллельному программированию Синхро, предназначенному для начального ознакомления с базовыми понятиями взаимодействия процессов и управления вычислениями. На этапе перехода к многопроцессорным архитектурам возрастает актуальность обучения параллельному программированию, что требует развития языково-информационной поддержки введения в программирование. Язык Синхро ориентирован на начальное обучение параллельному программированию школьников младших и средних классов в терминах управления взаимодействием роботов. Для иллюстрации методов представления программ, использующих параллелизм, приведены небольшие примеры. Препринт представляет интерес для системных 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 параллельного программирования.

Siberian Division of the Russian Academy of Sciences
A. P. Ershov Institute of Informatics Systems

L.V. Gorodnyaya

EDUCATIONAL PARALLEL PROGRAMMING LANGUAGE SYNCHRO

Preprint
180

Novosibirsk 2016

The work is devoted to Synchro — a parallel programming learning language. It is a language for learning the basic concepts of interaction processes and computing control. We describe educational programming languages, illustrated with short fragments of programs. The plan of a language for the basic learning of parallel programming is suggested. The main steps of an educational program are considered.

1. ВВЕДЕНИЕ

Пришло время изучать информатику и программирование, начиная с мира параллелизма. Сколь ни сложен этот мир, системе подготовки программистов предстоит его освоить и создать методику полноценного ознакомления с его не очевидными явлениями! Алгоритмы становятся параллельными, программы — многопоточными, исполнение кода — многопроцессорным, очередное действие может начать выполняться до завершения предыдущего, средства управления кроме логических значений используют события и многое другое [2].

Прежде всего, следует прояснить основные вопросы:

- насколько изменяется постановка задачи при переходе к параллельным алгоритмам?
- в какой мере при постановке задачи следует учитывать и выбирать модель параллельных вычислений?
- как обосновать и измерить выигрыш от трудозатрат на разработку параллельного алгоритма?

Тридцать лет назад инициатива академика А. П. Ершова по обучению школьников информатике несколько обидела профессионалов, полагавших, что программирование — это занятие для людей с высшим образованием. Теперь при оценке образовательного значения парадигм программирования признают фундаментальными функциональное, параллельное и императивно-процедурное. Возможно, опережающее освоение параллелизма не сразу получит признание. Но реализация такой идеи возможна на базе мультпарадигмальных учебных языков программирования [6].

Первая цель опережающего ознакомления с понятиями и явлениями параллелизма — профилактика жесткого привыкания к принципам традиционного последовательного императивно-процедурного программирования. Более широкое понимание программирования необходимо для специализации в области разработки распределенных информационных систем, а также приложений для суперкомпьютеров, многопроцессорных конфигураций и графических процессоров. Программирование и его техническая основа в последние десятилетия претерпели значительные изменения [1]. Актуальность обучения параллельному программированию требует развития языковой и системной поддержки учебных систем информатики, а их создание включает в себя разработку языка начального обучения параллельному программированию (ЯНОП), предоставляющего средства экспериментирования с основными моделями параллелизма.

Проект учебного языка Синхро нацелен на ознакомление с основными явлениями и моделями параллельного программирования, встречающимися в учебно-методических и научных материалах [9], языках высокого уровня (ЯВУ), языках высокопроизводительного программирования и при организации сетевых информационных сервисов.

2. ВЫБОР РЕШЕНИЙ

Сложность перехода от первичных навыков императивно-процедурного программирования к организации параллельных процессов в значительной мере обусловлена системой обучения, требующей все упорядочивать, выстраивать в однозначные, безальтернативные, последовательные построения, без выражения зависимости программируемых решений от динамики выбора структур данных и порядка действий по их обработке. Языки XXI века, как правило, поддерживают основные парадигмы программирования на ЯВУ — императивное, функциональное, логическое и объектно-ориентированное, что позволяет программировать решения задач с разным уровнем изученности в рамках единой языковой обстановки и получать навыки работы на всех фазах полного жизненного цикла программ на основе одной системы программирования.

Языки параллельного программирования занимают видное место среди функциональных языков. Одна из причин заключается в том, что функциональные программы свободны от побочных эффектов и ошибок, непредсказуемо зависящих от реального времени. Это существенно упрощает отладку программ. История языков программирования накопила ряд работоспособных идей по эффективной компактизации представления естественного параллелизма при серийной обработке сложных структур данных [5].

Для многих ЯВУ характерно включение в семантику языка одной конкретной модели параллелизма. Так, функциональный язык APL нацеливает на обобщение скалярных операций до обработки однородных векторов произвольной размерности. Algol-68 поддерживает управление процессами в терминах семафоров и критических участков. Язык Ada предлагает механизм «рандеву», требующий для обмена данными одновременного существования процессов. Функциональный язык Sisal предоставляет ряд средств формирования пространства итераций для параллельного исполнения циклов и для свертки полученных параллельно значений в общий результат. Интересен язык mpC, нацеленный на представление многопроцессорных конфигураций при выполнении многопоточных программ, допускающих

синхронизацию с помощью барьеров. Языки БАРС и Поляр используют сети управления процессами и дисциплиной доступа к данным. Производственные инструменты MPI и Open MP связаны определенными моделями параллелизма, поддержанными на уровне аппаратуры, что чревато трудоемкостью известной проблемы мобильности программ. Многие идеи организации процессов на уровне операционных систем ограничены понятиями о параллельной обработке на базе очередей и векторов [4].

Проблемы обучения параллельному программированию 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 и т. 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 [4]. Обычно учебные языки программирования обеспечивают быстрое изучение базовых понятий с демонстрацией первых успехов и облегчают переход от абстрактной алгоритмизации к практике отладки программ на производственных системах.

При проектировании предлагаемого здесь языка Синхро учтены идеи наиболее известных учебных языков программирования, таких как Basic, Pascal, Logo, Grow, Karel, Робик, A++, Oz. Особенно важен опыт методически обусловленного введения понятий программирования в языке начального обучения программированию Робик. Тщательная проработка понятий в этом языке может служить базисом для любой профориентации учебного процесса по информатике, программированию, информационным технологиям и допускает естественное развитие для перехода к представлению взаимодействия асинхронных процессов. Следует отметить, что изначально система понятий языка Робик содержала резерв для изучения параллельных программ. Программа могла включать и выключать разных исполнителей, обладающих своими системами команд, и назначать исполнителей отдельных действий [7]. Этот резерв не был востребован в конце 1970-х годов, но в наши дни он обретает актуальность.

Чуть позже, в конце 1980-х, увидел свет перевод на русский язык превосходной книги блестящего авторитета Т. Хоара «Взаимодействующие последовательные процессы» [9], в которой на простых проблемах управления автоматами показано, что параллельные композиции внешне не сложнее последовательных, если не акцентировать внимание на том, что

отладка параллельных процессов много сложнее, чем последовательных. Тем не менее, модель Хоара чувствительна к обнаружению достаточно тонких ошибок при создании программ. Поэтому здесь примеры Хоара дополнены рассмотрением вопросов отладки и методов минимизации ее объёма с помощью выделения типовых, многократно используемых фрагментов, реализуемых с учётом синтаксической правильности результата подстановки фрагментных переменных. Разработка программ для организации взаимодействия процессов отличается от подготовки обычных программ на весьма глубоком уровне, что можно показать на модели интерпретирующего автомата для языка управления процессами [5]. Такой автомат требует реализации дополнительной таблицы событий и структуры данных для очередей, регулирующих доступ к данным.

При определении реализации языка Синхро, использованы идеи языков Lisp, F#, C#, допускающих динамическую обработку кода программ. Это позволяет обеспечить лаконизм представления программ и упростить реализацию интерпретатора, компилятора и абстрактной машины. Следует отметить проблему расширяемости языка и системы программирования по мере развития средств и методов параллельных вычислений, обусловленную высоким темпом прогресса в области элементной базы и информационных технологий в целом. Многие понятия языка программирования — схемы управления программой, образующие программу действия, вычисления и данные — при переходе к параллельному программированию претерпевают изменения, и возникает необходимость в дополнительных понятиях. Основные понятия языка Синхро — фрагменты программ, образующие программу действия, вычисления и данные. Многие понятия при переходе к параллельному программированию претерпевают изменения, и возникает необходимость в дополнительных понятиях. На уровне языка Синхро предлагается следующая трактовка основных понятий:

Схема определяет варианты **управления** действиями в многопоточной **программе,** предназначена для параметризации барьеров, фрагментов и исполнителей потока.

Программа — это комплекс из **потоков**, выполняющийся в общей **памяти** — заранее заданном **контексте**, возможно с выделением значимых барьеров-**синхронизаторо**в. В последнем случае невыделенные барьеры не влияют на выполнение потоков. По умолчанию контекст содержит систему программирования на входном языке. Должен существовать контекст, в котором синхронизация потоков корректна.

Обстановка — представление контекста, содержащее перечень используемых объектов и исполнителей.

Поток — это комплекс из **фрагмент**ов, возможно синхронизованных по одноименным **барьер**ам с другими потоками. Должен существовать контекст, в котором выполнимы все действия потока.

Слой состоит из действий, порядок выполнения которых не задан, но допускает одновременное выполнение на разных процессорах.

Линия задает порядок выполнения действий как порядок их вхождения.

Обход дает возможность определять ветвящиеся процессы.

Включение обеспечивает использование общих фрагментов потока или схемы.

Назначение позволяет распределять работу программы по процессорам.

Итерация обеспечивает многократность выполнения потока.

Рецепт — хранимый вместе со спецификацией обстановки сценарий, ждущий своего времени.

Результативность вычислений можно повышать с помощью специально устроенных данных — так называемых «рецептов». Рецепт предназначен для отложенного исполнения фрагмента программы и представлен как пара из фрагмента и контекста для его выполнения, называемая замыканием фрагмента. Выполнение параллельных процессов часто требует независимости порядка вычислений от последовательности представления действий в программе. Общие решения по обеспечению лаконизма, конструктивности и расширяемости приводят к трансформационной семантике языка программирования, определяющей пространство допустимых преобразований программы.

Основные методы лаконичного представления массовых вычислений связаны с использованием неявных циклов, позволяющих избежать выписывания однотипных схем над стандартными структурами данных типа многомерных векторов. Так, например, результат операции над скалярами в языках параллельного программирования обычно распространен на произвольные однородные структуры данных. Список операций, допускающих распространение, определен реализацией. Обычно это арифметические операции (+ - * /). Этот механизм в Синхро распространяется на технику применения функций, что повышает лаконизм выражений. Кроме того, из бинарных операций можно конструировать фильтры. Результат фильтрации исчезает из аргумента — он переносится в другую структуру данных или сохраняется как значение. Структура из фильтров дает структуру из результатов их применения к одному и тому же аргументу.

При определении функций кроме обычных переменных используются фрагментные, с помощью которых можно формировать типизированные макроопределения. Редактирование фрагментов кроме простого сцепления

строк использует синтаксические макросы, вид параметров которых задаётся как синтаксическое подобие вхождению переменной в заданную строку согласно синтаксису используемого языка. Для выражения синтаксического подобия введена бинарная операция «~~», задающая вид первого аргумента с помощью строки, являющейся вторым аргументом. Выражение «v ~~ s v t» понимается как «переменная “v” имеет вид, заданный её вхождением в строку “s v t”». Это означает, что при подстановке “v” в “s v t” должна получаться правильная конструкция используемого языка. Правильность означает существование Pref и Suff, таких, что сцепление строк «Pref», «s», «v», «t» и «Suff» является допустимой программой в используемом языке. Все вхождения фрагментной переменной во фрагмент должны быть синтаксически эквивалентны её вхождению в строку, задающую вид переменной.

3. МНОГОПОТОЧНЫЕ ПРОГРАММЫ

Язык Синхро приспособлен к демонстрации небольших **многопоточных программ** с целью изучения основных явлений и моделей параллелизма методом показа типичных проблем организации параллельных процессов, заметных при выполнении простых этюдов в форме создания учебных игр. Язык не поддерживает статической иерархии определений и областей действия имён. Локализация имён используется лишь при определении исполнителей и функций.

Многопоточная программа представляется как **комплекс** потоков и может быть размечена на **слои**. Каждый поток состоит из ряда вычисляемых, возможно, помеченных **барьерами**, действий. Барьеры выполняют роль точек **синхронизации**, разбивающих программу на слои. Синхронизация потоков заключается в согласовании времени выполнения помеченных барьерами действий, образующих в программе слой. Каждый слой начинает выполняться одновременно и завершается до выполнения следующего слоя. Выражения одного слоя из разных потоков с одинаковой пометкой выполняются одновременно.

При выполнении многопоточной программы известно, из каких потоков она состоит и каков набор точек синхронизации потоков, разбитых на слои. Достижение барьера 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. Синхросети

позволяют независимые описания процессов связывать в терминах разметки барьерами. Узлы с одинаковой синхронизирующей разметкой срабатывают одновременно. Полное представление об асинхронных процессах, их эффективности и проблемах организации дают работы по сетям Петри.

Тексты программы формируются как **композиции действий и фрагментов**. Правило композиции влияет на упорядочение действий при их выполнении, возможно, отличающееся от порядка вхождения в сценарий. При отладке сценариев можно задавать условное время показа отдельных **действий**. При выполнении программы могут использоваться внутренние команды, недоступные для внешнего управления. Все команды допускают безусловное и **условное выполнение**. Последнее предназначено для эффективного реагирование на события и условия. Условное выполнение команды приводит к сбросу соответствующего сигнала. По мере выполнения действия формируются сообщения о **готовности к выполнению**, т. е. началу, о собственно выполнении и о **завершении действия**. Действия, связанные с изменением состояния памяти, подчинены механизму **транзакций**, т. е. признание их безуспешными влечёт **восстановление памяти** в состояние, предшествующее этому действию.

Ведущее понятие ЯНОП Синхро — «исполнитель», способный выполнять команды[1], причем исполнителей может быть много, и они могут обладать разными системами команд. Программы в Синхро строятся из потоков, а потоки — из действий, выполнение которых может быть обусловлено ожиданием времени (пауза) или другого сигнала, любым предикатом или вероятностью срабатывания. Последнее означает, что при исполнении ведется учет частоты выполнения вероятностных действий как в системах для разработки компьютерных игр. Кроме того, можно объявить планируемую длительность выполнения действий.

Постановка учебной задачи для экспериментов с параллельными алгоритмами на языке Синхро формулируется в терминах оперирования исполнителями, включая автоматизацию взаимодействующих процессов с помощью мультипрограмм, определяющих их работу как автоматов [3]. Кроме обычных команд из меню допустимых команд исполнителя и средств синхронизации процессов в терминах барьеров — событий, происходящих в разных процессах одновременно, в языке Синхро есть специальные метакоманды для управления составом исполнителей и их редактирования при формировании контекста исполнения программы. При организации слож-

[1] Унаследовано от языка Робик и школьного курса информатики. Используется как общий синоним терминов «автомат», «процессор», «робот».

ных данных и действий используются общие структуры или средства композиции, такие как списки, вектора и множества, обеспечивающие представление отношений «после», «одновременно» и «взаимоисключено», причём последовательность вычисления компонентов может не зависеть от порядка их размещения в программе или в памяти. Механизмы применения функций и операций рассматриваются как мета-операции, допускающие просачивание.

Действия в языке Синхро приспособлены к варьированию дисциплины доступа к данным и схемы управления процессами обработки комплексов с помощью схемы обхода, выглядящей подобно оператору IF без ELSE. Потоки могут отлаживаться и выполняться независимо друг от друга в предположении, что каждый из них можно располагать отдельно.

В случае многопоточных программ возможно преобразование потоков, нацеленное на сведение к однородной системе, однозначно отображаемой на заданный комплекс процессоров размещением потоков по процессам или назначением процессоров для выполнения потоков.

Такое требование можно выразить формулой:

[(b0: ; b1: ; … bK:), p1!(b0: d0i; …), … pN!(b0: d0j; ...)], где i и j обозначают принадлежность потоку.
, — параллельное или одновременное исполнение,
; — последовательное исполнение,
! — назначение процессора,
b0:, b1:, … — барьеры,
p1, pN, … — процессоры,
d0i, d0j, … — действия,
(b0:; b1:; … bK:) — шкала событий/барьеров,
pM!(b0: d0i; …) — программа (последовательность) действий для процессора pM.

В такой «причесанной» форме все потоки начинаются с барьеров, и общая шкала событий упорядочена так, что последовательность событий потока ей не противоречит. Можно считать, что процессоры включаются сами. Шкала событий содержит списки ожидающих потоков. Действия, выполняемые процессорами, соотнесены с их исходными потоками.

Достаточно простые преобразования сети потоков позволяют варьировать схемы потоков и многие конструкции языка программирования сводить к взаимодействию простых потоков:

(A;B) ↔ (a:A; b:B) #– Расстановка — стирание барьеров.

(a:A; b:B) ↔ [[a:A, b:B], (a: ; b:)] — вынесение последовательного управления в отдельный поток — восстановление последовательности действий.

(a: A; b: B) ↔ [(a: A; b:), (b: B)] — разрез последовательности — перенос «хвоста», если нет локальной информационной зависимости между A и B.

(a: A; b: B) ↔ [a: A, b: B] — разбиение последовательности на потоки — сплющивание линии в слой, если A и B информационно не связаны.

[a: A, b: B] ↔ { (a: A; b: B) | (b: B; a: A)} — слияние потоков в одну последовательность — вытягивание слоя в линию. Выбор варианта требует учета последовательности барьеров в других потоках и общей шкале событий. Критерий оптимальности — объем выполнимых вычислений.

((a: A; b: B) ; c: C) ↔ (a: A; (b: B ; c: C)) ↔ (a: A; b: B ; c: C)

[[a: A, b: B] , c: C] ↔ [a: A, [b: B , c: C]] ↔ [a: A, b: B, c: C]

([A,B];C) ↔ [(A;C), (B;C)] — исключение слияний — слияние совпадающих продолжений.

[(A;B),C]↔ ([A,C]; [B,C]) — распределение параллельного потока — слияние совпадающих потоков

[[a: A], [a: B]]↔ [a: [A, B]] — варьирование числа одновременных потоков.

Обратимость преобразований и чувствительность их результата к информационным связям между фрагментами программы требуют формализации критериев применимости трансформаций и выбора подходящего варианта. Можно констатировать, что выяснение информационной связанности действий B и A сводится к проверке существования контекста, в котором различны результаты программы C при изменении порядка вычислений B и A.

$$[(A; B), C] \neq [(B; A), C]$$

Из числа базовых средств можно вывести ветвления, циклы и вызовы функций, реализуя их средствами синхронизации потоков.

При отладке формируется ряд контекстов, демонстрирующих конкретные свойства фрагментов, из которых собирается полная программа. Это контексты для отдельных потоков, для пар синхронизованных потоков, для интегрированной из потоков программы, а кроме того, контексты для удостоверения наличия-отсутствия информационных связей между фрагментами. Возможны пользовательские преобразования схем управления про-

цессами, что позволит не только минимизировать ручную аранжировку распараллеливаемых программ, но и даст основу для формирования библиотек преобразования схем программ.

4. УЧЕБНАЯ МЕТАФОРА ЯЗЫКА ПРОГРАММИРОВАНИЯ

Описание учебной версии языка использует метафору «фабрика разнотипных роботов для конструирования программно-управляемых игрушек». По этой метафоре конструктор игрушки строит определённую **обстановку** для комплекса взаимодействующих **роботов**.

Любой робот может выполнять небольшой набор общих команд языка Синхро. Команда реализуется как микроробот, обрабатывающий только простые данные. Команды могут быть внешними, доступными как кнопки оперирования, или специальными, внутренними, используемыми в сценариях при автоматизации поведения роботов. Каждый робот имеет свои команды ПУСК, ШАГ и СТОП. Разные **типы роботов** могут обладать специальной **системой команд**, включая регистры — это переменные и шкала **сигналов о событиях и условиях**, на которые робот может реагировать по ходу игры. Базовый механизм **взаимодействия роботов** представляется как **объявление и ожидание событий**. Конкретный робот может иметь своё **имя** и один или несколько своих **сценариев** поведения, задающих спектр допустимых команд. При включении каждого робота в обстановку появляются две кнопки управления этим роботом: **пошаговое оперирование** и пуск **автоматически выполняемого сценария**. Конструктор игры или представитель сервисной службы может настраивать Роботов и необходимый для их работы **инвентарь** на **стартовое состояние** до начала игры при формировании обстановки. Инвентарь отличается от робота отсутствием кнопок управления.

При создании игрушки конструктор может сценарии робота уточнить, включая изменение системы команд. Можно фиксировать изменение системы команд робота, реализуемое фабрикой как запуск производства нового типа роботов. Обстановка выполнения учебно-игровых программ устроена как МакроРобот, который схемоподобен роботу, только роль команд выполняют вовлечённые в игру роботы, а роль данных — **фрагменты** сценариев, допускающие редактирование, и инвентарь, имеющий фиксированное число состояний памяти, переключение между которыми подчинено определённой **дисциплине**. Работа МакроРобота похожа на работу препроцессора в производственных системах программирования. Специальные команды и операции МакроРобота позволяют воздействовать на всех

включенных роботов и редактировать фрагменты обстановки, кроме своих команд, своего сценария, а также команд и сценария МикроРобота. Простейшие действия — это выражения и команды. Более сложные действия строятся как композиции, использующие фрагменты и операции редактирования, выполняемые при формировании обстановки.

Робота сопровождает легенда, поясняющая в какой обстановке робот действует и каков смысл его системы команд. Кроме того, известно его изображение и вид кнопок управления роботом. Робот может обладать встроенным сценарием – программой автоматизации его поведения. Игра управляется многопоточной программой, выполняемой комплексом роботов, включая МакроРобота.

Допускается, что каждый поток выполняется отдельным роботом. **Связи** между роботами и фрагментами программы задаются специальной операцией указания роботу выполнить данный фрагмент, что допускает участие ряда роботов при выполнении одного потока. Роботы могут быть различны по системе команд и другим характеристикам. В таком случае необходимо, чтобы представление потока соответствовало системе команд робота и комплекту используемого инвентаря, точнее сценарий выполним заданным роботом. **Спецификация** робота должна хранить нужные характеристики, например, относительное время выполнения действий.

Каждый робот имеет встроенный самописец, регистрирующий ход процесса выполнения его части многопоточной программы в протоколе. Роботы генерируют протокол, внося в него сообщения о статусе выполняемых команд. Безусловные команды вносят в протокол события Н В К, условные команды – Н В К или Н К в зависимости от истинности **ключевого сигнала.** МакроРобот после редактирования сценариев и формирования обстановки вносит в протокол имена включаемых роботов, объявляемые ими события и готовность к **реагированию на события**. Кроме того, попутно собирается статистика **частоты выбора вероятностных ветвей**. Вероятности приводятся к общему знаменателю.

5. КАТЕГОРИИ УЧЕБНЫХ ЗАДАЧ И ТИПЫ РОБОТОВ

При подготовке примеров для демонстрации учебного языка параллельного программирования были приняты следующее ограничения:

- взаимодействия потоков выполняются только через синхронизацию;
- одновременно исполняемые потоки могут обмениваться данными через общую память;

– при взаимоисключении каждый поток работает в своей копии контекста;
– поддерживается список для восстановления многократно выполняемых фрагментов.

Для оперирования роботами выделяются команды уровня МикроРобота, устроенные как функции без параметров над простыми данными из общей памяти.

Работа команд может быть достаточно точно определена в стиле абстрактной или виртуальной машины как система переходов над четырьмя простыми регистрами: П С У Д → П' С' У' Д', используя следующие обозначения:

П — протокол или результат работы команды,
С — обозначение выполняемой команды,
У — управляющий сигнал, разрешающий выполнять команду,
Д — данное, используемое при выполнении команды.

В качестве примера рассмотрим механизм работы кнопки для команды «К».

Кнопка К: П С У Д → П' С' У' Д'

Нажатие кнопки К приводит к появлении 1 в регистре К.У, что дает команде право выполнения.

_ ком 1 Д → ком(Д) ком 0 Д

Робот.П += В_К += ком(Д)
\\ робот до команды пишет Н_К, после — З_К.

Робот.П += «З_К»

Аналогично можно описать как робота работу обычной переменной, которую объявляют, инициируют, читают и обновляют её значение. Команды работают на общих регистрах МикроРобота.

Переменная V: П С У Д → П' С' У' Д'

→: **(**_ → 1 имя) → **(**П' → 0 имя)
Р.Д += имя
Робот.П += В_К += ком(Д)
\\ робот до команды пишет Н_К, после — З_К.

←: **(**_ ← 1 имя) → (П' ← 0 имя)
Р.Д -= имя

Робот.П += В_К += ком(Д)
\\ робот до команды пишет Н_К, после — З_К.

↑ !: (_ ! 1 имя) → (Р.Д.имя ! У0 имя)
Робот.П += В_К += ком(Д)
\\ робот до команды пишет Н_К, после — З_К.

\\ перепись значения в протокол команды присваивания
_ : (_ _ 1 **знач**) → (**знач** _ 0 **знач**)
\\ команды работают на общих регистрах
↓ := : (знач С У имя) → (П' С' У' Д')
Р.Д.имя = знач

Робот.П += В_К += ком(Д)
\\ робот до команды пишет Н_К, после — З_К.

Реквизит исполнителя или инвентарь робота приспособлен к установке стартовых сигналов МакроРоботом, инициированию данных и их изменению роботами, входящими в спецификацию инвентаря.

6. ПРИМЕРЫ ЗАДАЧ С РЕШЕНИЯМИ

Показать и пронаблюдать общие модели параллельных вычислений и связанные с их применением явления можно на задачах, описанных в книге Хоара [9]. Используется материал олимпиадных задач и учебных примеров из школьных и факультативных курсов информатики. Кроме того, подобраны задачи для демонстрации конкретных проблем организации взаимодействия процессов, таких как «состояние гонки» и других «плавающих» ошибок, проявляющихся в случайные моменты времени и при изменении границ доступа к данным, «пропадающих» при попытке их локализовать. Решения учебных задач представлены на языке Синхро, наследующем конструкции языков Grow и Робик с учетом современных технологий разработки информационных систем.

1. Вводные примеры из книги Хоара

Пример 1. Простой торговый автомат — ПТА, способный принимать монетки и за них выдавать шоколадки. Описать сценарий управления автоматом ПТА, принимающим монету, а потом выдающим шоколадку.

Фрагмент	*Комментарий*
(! мон **;** ? шок **)**	\\ *сначала прием монетки;* \\ *потом выдача шоколадки*

Круглые скобки и «**;**» указывают, что это поток действий.
Команды ! и ? универсальны, доступны в любом автомате. Переменные «мон» и «шок» описаны внутри ПТА и доступны в его сценарии.

Пример 2. Простой торговый автомат может работать с доверием. Описать сценарий управления автоматом ПТА, допускающим выдачу шоколадки до получения монетки — доверяя покупателю.

Фрагмент	*Комментарий*
[! мон , ? шок **]**	\\ *прием монетки и независимо,* \\ *выдача шоколадки в любом порядке*

Квадратные скобки и запятая указывают, что это слой из независимых действий. Можно взять шоколадку, а потом дать монетку или наоборот.

Пример 3. Простой торговый автомат, допускает переход от управления автоматом к программе с учетом числа шоколадок в автомате. Автомат может по разным причинам не выдать шоколадку, например, из-за отсутствия. Пусть обстановка в N хранит число имеющихся в автомате шоколадок. Описать сценарий для автомата ПТА, сообщающий покупателю об отсутствии товара и возвращающего монету.

Фрагмент	*Комментарий*
n = 4; (! мон **;** [ЕСЛИ n ТО [? шок **,** n = n - 1] **,** ЕСЛИ_НЕ n ТО ? мон **]** **)**	\\ *N - число шоколадок в автомате* \\ *прием монетки* \\ *условие готовности автомата* \\ *выдача шоколадки* \\ *учет остатка в контексте* \\ *возврат монетки*

Задано, что в ПТА хранится 4 шоколадки.
После приёма монетки включается слой из действий, допускающих обход:
при наличии шоколадок запускается выдача шоколадки и
пересчёт числа оставшихся шоколадок;

при отсутствии шоколадок возвращается монетка.
Операции «=» и «-» универсальны, т. е. общедоступны.

Пример 4. Простой торговый автомат может допускать многократное пополнение запасов шоколадок. Описать сценарий автомата ПТА, который после исчерпания запасов принимает пополнение и снова работает.

Фрагмент	*Комментарий*
пта_ф = (n) (! мон **;** [ЕСЛИ n ТО [? шок **,** пта_ф (n - 1)] **,** ЕСЛИ_НЕ n ТО (? мон **; ? «шоколадок пока нет»)** **]** **)**	\\ *число шоколадок в автомате* \\ *прием монетки* \\ *условие готовности автомата* \\ *выдача шоколадки* \\ *учет остатка* \\ *возврат монетки*
(пта_ф (4) ; пта_ф (20) ; пта_ф = (100))	*запуск автомата ПТА,* *последовательно выдающего* \\ *4, 20 и 100 шоколадок.*

Определение функции от числа, заданной фрагментом из Примера 3, учитывающим наличие шоколадок.

Пример 5. Простой торговый автомат ПТА_N_M, выполняющий подсчет выручки. Считаем, что монетоприемник по объему превышает объем возможной выручки.

Фрагмент	*Комментарий*
пта_фв = (n m) (! мон **;** [ЕСЛИ n ТО [? шок **,** пта_фв (n - 1, m + 1)] **,** ЕСЛИ_НЕ n ТО (? мон **; ? «шоколадок пока нет»** **;** РЕЗУЛЬТАТ (m - 1) **)** **]** **)**	*M – число монет в автомате* \\ *прием монетки* \\ *условие готовности автомата* \\ *выдача шоколадки* \\ *учет остатка* \\ *и подсчет выручки* \\ *возврат монетки* \\ *учёт возврата*
в = 0; в = пта_фв (4, в) ;	*Подсчёт выручки при продаже*

в = в + пта_фв (20, в) ; в = в + пта_фв = (100, в) **)**	\\ *4, 20 и 100 шоколадок.*

Определение функции от числа шоколадок и накопленной выручки.

Пример 6. Счетчики (локализация функции)

Фрагмент	*Комментарий*
Учет = (n) { ЕСЛИ n ТО Учет (n - 1) }	\\ *Выделение из примеров 4 и 5* \\ *функции учёта шоколадок*
Выручка = (n m) [ЕСЛИ n ТО Выручка (m + 1) **,** ЕСЛИ_НЕ n ТО РЕЗУЛЬТАТ (m - 1) **]**	\\ *Выделение из примера 5* \\ *функции подсчёта выручки*

Пример 7. Взаимодействие процессов. Расстановка барьеров для взаимодействия со счётчиком. Имеются описания разных счетчиков, контролирующих работу автомата. Надо синхронизовать действия счетчиков.

Фрагмент	*Комментарий*
пта_б = () (! Мон ; ? Шок ; пта_б)	\\ *зацикливание*

Бесконечный процесс обслуживания

Пример 8. Взаимодействие процессов. Расстановка барьеров для управления процессом.

Фрагмент	*Комментарий*
учет_б = (n) [ЕСЛИ n ТО **Контроль:** учет_б (n - 1)]	
пта_б = ()	

(! Мон ; ? Шок ; **Контроль** : пта_б)	

Разметка фрагментов из примеров 6 и 7 барьерами для синхронизации процессов учета числа шоколадок и шагов обслуживания.

Пример 9. Определение взаимодействия процесса со счётчиком. Расстановка барьеров для синхронизации процессов.

Фрагмент	*Комментарий*
пта_и_учет = (n) **[Контроль] [** пта_б **,** учет_б (n) **]**	\\ *N шагов рекурсии* \\ *общий барьер*

Барьер в рекурсивной функции УЧЕТ_Б и бесконечном процессе ПТА_Б ограничивает число выполнимых шагов

Пример 10. Автомат с заманиванием

Фрагмент	*Комментарий*
пта_б = () (! Мон ; **Контроль** : ? Шок ; пта_б)	\\ *Бесконечный процесс,* \\ *выдающий шоколадки*
подарок = () (**Контроль:** ВЕРОЯТНО ИНОГДА % ? «**возьми подарок**» ; подарок)	*Бесконечный процесс,* \\ *иногда выдающий подарок*
пта_и_подарок = () **[Контроль] [** пта_б **,** подарок **]**	\\ *N шагов рекурсии* \\ *общий барьер*

Бесконечный процесс, совмещающий выдачу шоколадок с эпизодической выдачей подарка

Пример 11. Настройка автомата с заманиванием на подсчёт числа выданных призов

Фрагмент	*Комментарий*
пта_б = () (! Мон ; **Контроль** : ? Шок ; пта_б)	
Выручка = (m) [**Контр_В:** Выручка (m + 1) **,** ЕСЛИ_НЕ n ТО РЕЗУЛЬТАТ (m - 1) **]**	\\ *Барьер, синхронизующий* \\ *выдачу приза* \\ *с подсчётом их числа*
пта_учёт_и_подарок= (n) **[Контроль , Контр_В] [** пта_б , Выручка(0), учет_б (n) , подарок **]**	\\ ***N шагов рекурсии*** \\ *Общие барьеры* \\ *в разных потоках*

Пример 12. Варианты взаимодействий

Фрагмент	*Комментарий*
пта_учет_выр = (n_шок n_мон) [Контроль] [пта_б () , учет_б (n_шок), выручка (n_мон)] **пта_с_учетом =** (n_шок) **[контроль] [** пта_б **,** учет (n_шок) **]**	\\ *барьер во всех процессах* \\ *автомат с учетом числа* \\ *шоколадок и монет* \\ *N шагов рекурсии* \\ ***общий барьер***

Пример 13. Варианты взаимодействий

Фрагмент	*Комментарий*
пта_учет_дар = (n_шок) **[** контроль] [пта_б () , учет_б (n_шок)] подарок **]**	\\ *процессы синхронизованы* \\ *процесс без барьеров* **\\ *автомат с учетом шоколадок,*** **\\ *возможно заманивание***

2. Макро Робот. Редактирование фрагментов

Редактирование фрагментов кроме простого сцепления фрагментов использует синтаксические макросы, вид параметров которых задаётся как синтаксическое подобие вхождению переменной в заданную строку согласно синтаксису используемого языка сценариев.

Пример 14. Функции (параметризация действий с заданием реквизита исполнителей)

Фрагмент	*Комментарий*
ф_пта = МАКРО (м ~~ “!м” ш ~~ “ ?ш ”) **(** !м **;** ?ш **; Контроль:** ф_пта **)**	\\ *по умолчанию параметры сохраняются*

Параметры для вставки в синтаксически подобный фрагмент

Пример 15. Функции (параметризация действий с заданием реквизита исполнителей).

Фрагмент	*Комментарий*
ф_пта (@жетон, @игрушка)	\\ *явная параметризация отложенных действий*

другой автомат, сценарий которого содержит синтаксически подобные фрагменты

Пример 16. Функции (параметризация действий с заданием реквизита исполнителей). Вызов исполнителя.

Фрагмент	*Комментарий*
[Контроль**]** **ИТА !! [**ф_пта (@жетон, @игрушка) **,** выручка (0) **]**	\\ *явное задание исполнителя*

Пример 17. Управление рекурсией

Фрагмент	*Комментарий*
учет **= (** n **)** ЕСЛИ n ТО **Контр:** учет_б (n – 1)	
ф_пта = (@мон @шок) **(** !мон **;** ?шок **;** **Контр:** ф_пта **)**	
[Контр] **[** ф_пта (жетон, игрушка) **,** учет (4) **]**	\\ *выдать 4 игрушки*

Пример 18. Параметр — барьер

Фрагмент	*Комментарий*
учет_б **= (** n **b ~~ “ b: ”)** учёт = (n) ЕСЛИ n ТО **b:** учет_б (n – 1) учет_б (10 контр)	\\ *параметр-барьер*

Пример 19. Переименование барьера и реквизита

Фрагмент	*Комментарий*
ф_пта_б = (м ~~ “!м”, ш ~~ “ ?ш ”, **р ~~ “ b: ”**) **(!** м **;** **?** ш **;** **р:** ф_пта_б **)**	\\ *параметры — реквизит и барьер*

Пример 20. Синхронизация параметризованными барьерами

Фрагмент	*Комментарий*
[Контроль] **[** ф_пта_б (@жетон, @игрушка, **Контроль:**) **,** учет_б (4, **Контроль:**)**]**	\\ *подстановка барьера* \\ *в автомате 4 игрушки*

3. Декомпозиция автоматов

Пример 21. Разложить простой автомат ПТА на два более простых: принимающий монету и выдающий шоколадку.

Фрагмент	*Комментарий*
[роб_мон !! (!мон; к:) ; роб_шок !! (к: ? шок)]	\\ ***(*** *! мон ; ? шок* ***)*** \\ *сначала прием монетки,* \\ *потом выдача шоколадки*

Сначала один робот принимает монетки, а другой выдаёт шоколадки

Пример 22. Простой торговый автомат с доверием разложить на два автомат: один автомат допускает выдачу шоколадки до получения монетки другим автоматом, доверяя покупателю.

Фрагмент	*Комментарий*
[роб_мон !! [!мон; к:] , роб_шок !! [к: ? шок]]	\\ **[** ! мон **,** ? шок **]** \\ *прием монетки и независимо* \\ *выдача шоколадки в любом порядке*

Один робот принимает монетки, а другой выдаёт шоколадки в произвольном порядке

Пример 23. Определение робота, работающего как простой автомат ПТА, принимающий монету, а затем выдающий шоколадку.

Фрагмент	*Комментарий*
Робот_пта = { ИМЕНА мон , шок ; КНОПКА дай ! мон ; КНОПКА возьми ? шок ; [ ] **(** ! мон **;** ? шок **)** }	\\ *Объявление робота* \\ *Описание имён* \\ *кнопка приёма монеток* \\ *кнопка выдачи шоколадок* \\ *без барьеров* \\ *сценарий допустимых действий:* \\ *сначала прием монетки,* \\ *потом выдача шоколадки*

Пример 24. Определение обстановки для игры, предоставляющей оперирование роботом, работающим как простой автомат ПТА, принимающий монету, а затем выдающий шоколадку.

Фрагмент	*Комментарий*
{ Робот_пта , КНОПКА дай ; КНОПКА возьми ; КНОПКА шаг ; [ ] **[** дай **,** возьми **]**	\\ *Встраивание робота* \\ *кнопка приёма монеток* \\ *кнопка выдачи шоколадок* \\ *кнопка оперирования роботом* \\ *без синхронизации* \\ *сценарий допустимых нажатий:* \\ *прием монетки,* \\ *выдача шоколадки*

4. Олимпиадная задача для младших школьников

Пример 25. 2 сковороды — 3 котлеты (производительность)[2]:

Написать программу жарки 3-х котлет на двух сковородах такую, что сковороды не простаивают.
3 котлеты на 2 сковороды.
Для обжарки одной стороны котлеты требуется 5 минут. Имеется 2 сковороды, на них надо как можно скорее поджарить 3 котлеты.
Надо предложить и ясно записать масштабируемое решение.

Фрагмент	*Комментарий*
Обж_3_2 = [ ] **{** сковороды = [1..2 :] ; Обж = Функция (Жарить Готово) [ЕСЛИ_НЕ Жарить ТО РЕЗУЛЬТАТ готово **,** ЕСЛИ Жарить ТО **(** жарить → сковороды ; ЖДУ 5 ; сковороды = (сковороды **-** 1) ; (сковороды ?≠0) → жарить **;**	\\ *без барьеров* \\ *Имеется 2 пустых сковороды для жарки* \\ *оба параметра - очереди* \\ *= ((2 2 2) ())* \\ *рекурсия работает* \\ *до опустошения очереди* \\ *в вектор из очереди переданы* \\ *два элемента* \\ *переданные элементы* \\ *исчезают из очереди* \\ *Время «жарки»* \\ *Одна сторона обжарена* \\ *Разделяем котлеты на готовые*

[2] на Intel-семинаре в Москве упоминалась как пример немасштабируемого алгоритма.

(сковороды ?=0) → готово **;** Обж (жарить **,** готово) **)**]	\\ *и требующие жарки* \\ *«Недожаренные» идут в очередь* \\ *на жарку* \\ *«Готовые» - в тарелку (со 2-го витка)*
Обж ((2 2 2) ( )) **}**	\\ *Надо обжарить 2 стороны 3-х котлет.* \\ *Дана пустая тарелка* \\ *для готовых котлет*

Скобки можно опускать, если это не мешает распознаванию выражения.

5. Задачи из описаний ЯП

Пример 26. Однородные серии автоматов. Быстрая сортировка (часто приводится в описаниях языков параллельного программирования).

Фрагмент	*Комментарий*
ФУНКЦИЯ qs (Data) [ЕСЛИ (size (Data) < 2 ТО Data) , КРОМЕ (size (Data) < 2 (Pivot = Data [1] ; Low, Mid, High = Data ((?<, ?=, ?<) Pivot) ; (qs (Low) || Mid || qs (High)))]	\\ *просачивание* \\ *по списку фильтров* \\ *и структуре*

Пример 27. Однородные серии автоматов. Параллельный «пузырёк»

Фрагмент	*Комментарий*
числа A [1 .. 2K]; процессоры П [1 .. K] ; ЦИКЛ П [1 .. K] **!!** (ФУНКЦИЯ (i) ЕСЛИ A [2*i - 1] < A [2*i] ТО A [2*i - 1] <=> A [2*i]}	\\ *просачивание по процессорам* \\ *Вызов исполнителей !* \\ *Вектор процессоров вызывает:* \\ *Функцию процессора на* \\ *окрестности (2*i):* \\ *ЕСЛИ левое число* \\ *меньше (2*i)-го,* \\ *ТО они меняются местами;*

; \\ затем ЕСЛИ A [2*i + 1] > A [2*i] ТО A [2*i] <=> A [2*i +1]) [1 .. K] \\ просачивание ПОКА ВЫПОЛНЯЛОСЬ (<=>)	\\ *затем работа с правым* \\ *от (2*i) числом:* \\ *ЕСЛИ (2*i)-ое число* \\ *меньше правого,* \\ *ТО они меняются местами;* \\ *просачиваемую* \\ *по диапазону 1 .. K .* \\ Выход при отсутствии \\ перестановок.

ЗАКЛЮЧЕНИЕ

Рассматривая задачу формализации языков параллельного программирования как путь к решению проблемы адаптации программ к различным особенностям используемых многопроцессорных комплексов и многоядерных процессоров, следует видеть, что решение этой проблемы кроме ознакомления с проблемами и методами требует разработки новых методов реализации программ с акцентом на тестирование, верификацию и отладку, а также развития средств и методов ясного описания семантики языков параллельного программирования, включая представление программируемых преобразований текста и переносимого кода программы с удостоверением их корректности [8].

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 и решений в смежных областях представлен в [5, 6].

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.

СПИСОК ЛИТЕРАТУРЫ

1. 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.
2. Воеводин В. В., Воеводин Вл. В. Параллельные вычисления. — СПб.: БХВ-Петербург, 2002. — 608 с.
3. Городняя Л.В. О курсе «Начала параллелизм» // Ершовская конференция по информатике. Секция «Информатика образования». — Новосибирск: ИСИ СО РАН, 2011. — С. 51–54.
4. Городняя Л.В. О проблеме автоматизации параллельного программирования // Тр. Междунар. суперкомпьютерной конф. «Научный сервис в сети Интернет: многообразие суперкомпьютерных миров» — URL: http://agora.guru.ru/abrau2014 .
5. Городняя Л.В. Парадигмы программирования. Часть 4. Параллельное программирование. — Новосибирск, 2015. — 74 с. — (Препр. / ИСИ СО РАН; № 175). — URL: http://www.iis.nsk.su/files/preprints/gorodnyaya_175.pdf.
6. 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
7. Звенигородский Г.А. Первые уроки программирования. — Библиотечка «Кванта». — М.: Наука, 1985. — Т. 41.
8. Степанов Г.Г. Пути обеспечения переносимости программ и опыт использования системы СИГМА // Трансляция и преобразование программ. — Новосибирск: ВЦ СО АН СССР, 1984. — С. 5–13.
9. Хоар Ч. Взаимодействующие последовательные процессы. — М.: Мир, 1989. — 264 с.

Л.В. Городняя

ЯЗЫК ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ СИНХРО, ПРЕДНАЗНАЧЕНЫЙ ДЛЯ ОБУЧЕНИЯ

Препринт
180

Рукопись поступила в редакцию 17.11.2016
Редактор Т. М. Бульонкова
Рецензент Ф.А. Мурзин

Подписано в печать 19.12.2016
Формат бумаги 60 × 84 1/16 Объем 1.71 уч.-изд.л., 1.88 п.л.
Тираж 60 экз.

Типография Оригинал-2, г. Бердск, ул. Олега Кошевого, 6, оф. 2
тел./факс: 8 (383) 328-32-38, (38341) 2-12-42, сот.: 8 913 987 77 67