# Моделирование и программирование

УДК 004.43 (042.4)
doi:10.38028/ESI.2022.25.1.009

## Место функционального программирования в организации параллельных вычислений

**Городняя Лидия Васильевна**
Институт систем информатики СО РАН,
Россия, Новосибирск, *gorod@iis.nsk.su*

**Аннотация.** Статья посвящена результатам анализа современных тенденций функционального программирования, рассматриваемого как метапарадигма решения проблем организации параллельных вычислений и многопоточных программ для многопроцессорных комплексов и распределённых систем. Учитывая мультипарадигмальность языков параллельного программирования, здесь использован парадигмальный анализ, позволивший выявить ряд особенностей и прогнозировать ход процессов применения программ, а также их изучения и разработки. Есть основания рассчитывать, что функциональное программирование способно повышать производительность программ с помощью предварительной подготовки их прототипов. Приведено описание семантических (универсальность, самоприменимость, равноправие параметров) и прагматических (гибкость ограничений, неизменяемость данных, строгость результата) принципов функционального программирования, а также следствий из этих принципов, нацеленных на снижение трудоёмкости и повышение технологичности создания прототипов, автономно развиваемых модулей и отладки программ (конструктивность, верификация, факторизация, непрерывность процессов, обратимость действий, унарные функции). Отмечена роль парадигмальной декомпозиции программ в технологии разработки долгоживущих программ, какими нередко бывают программы параллельных вычислений. Особо подчёркнута перспектива функционального программирования (ФП) как вспомогательной универсальной метапарадигмы решения сложных и важных задач, обременённых трудно удостоверяемыми и слабо совместимыми требованиями эффективности, надёжности, безопасности и улучшаемости. Показано разнообразие парадигмальных характеристик, присущих подготовке и отладке долгоживущих программ параллельных вычислений.

**Ключевые слова:** функциональное программирование, параллельные вычисления, языки программирования, парадигмы программирования, мультипарадигмальность

**Цитирование:** Городняя Л. В. Место функционального программирования в организации параллельных вычислений // Информационные и математические технологии в науке и управлении. – 2022. – № 1 (25). – С. 102-119. – DOI:10.38028/ESI.2022.25.1.009.

**Введение.** Функциональное программирование - одна из первых парадигм, нацеленных не просто на получение эффективной реализации заранее подготовленных алгоритмов, а на решение новых задач информационной обработки, обладающих исследовательским компонентом [1, 2]. Если множество языков программирования стремительно растёт, то число парадигм не столь велико. Оно допускает достаточно полное сравнение и выбор характеристик для чёткого разделения парадигм, уяснения причин их разнообразия и сферы

их успешного применения [3, 4]. Рассматривая результаты анализа проблем, средств и методов организации параллельных вычислений, а также подготовки многопоточных программ для многопроцессорных комплексов и распределённых систем, можно обратить внимание на явно широкий спектр постановок задач и соответствующих им приоритетов в принятии решений на разных этапах разработки и отладки программ. В результате методика парадигмального анализа позволяет сводить постановки решаемых задач к комплектам автономно развиваемых подзадач, оценивать их сходство и различия, учёт которых необходим при прогнозировании сложности процессов применения программируемых решений, начиная с планирования, изучения и организации разработки программ решения таких подзадач для долгоживущих программ, какими часто оказываются программы параллельных вычислений [5].

Изложение начинается с рассмотрения ряда парадигм параллельных вычислений, поддержанных в известных языках программирования (ЯП). Затем идёт обсуждение основных принципов и ограничений, обнаруживающихся при переносе техники ФП на параллельные вычисления. В заключении описаны требования к языковой поддержке параллельного программирования.

**1. Общее представление.** Рассматривая идеи функционального программирования и практику их применения при анализе проблем, средств и методов организации параллельных вычислений, следует обратить внимание, что уже созданы значительное число языков и систем параллельного программирования. Используя систематизацию парадигм программирования на основе различий в приоритетах принятия программируемых решений, можно сделать вывод, что одной из причин сложности разработки программ параллельных вычислений является их скрытая мультипарадигмальность [6]. Для разработки параллельной программы многое требуется продумывать по-разному одновременно в различных парадигмах, удобных для решения отдельных подзадач без традиции и навыков решения полного комплекса подзадач в единой обстановке. Наиболее очевидно парадигмальные различия видны при решении задач масштабирования вычислений на различные многопроцессорные комплексы, синхронизации взаимодействий над локальной и общей памятью в многопоточных программах, представлении естественного асинхронного параллелизма уровня постановок задач и достижения высокой производительности программ с учётом критерия полноты загрузки доступных много-процессорных комплексов или распределённых систем. Отдельно имеет место образовательная проблема – ознакомление с феноменами параллелизма, дающее специалистам интуитивное понимание методов решения особо трудных задач. Новые работы по ФП существенно нацелены на поиск более производительных решений проблем параллельного программирования [7]. Созданы заметное число языков и систем программирования (ЯиСП), позволяющих решать отдельные из этих задач в рамках конкретных парадигм, поддержка которых представлена в разных языках программирования (табл. 1).

**Таблица 1.** Проблемы ПВ, поддержанные в разных языках программирования и инструментах

| *№* | *Проблема* | *Парадигма* | *ЯП и API* |
|---|---|---|---|
| 1 | Масштабирование | Многопроцессорное программирование | VHDL, XC, СИГМА, bash, Occam, mpC, Эль-76, Limbo, Kotlin, MPI |
| 2 | Синхронизация потоков | Синхронное (синхронизующее) программирование | APL, VAL, Sisal, Alef, E, X10, LuNA, Charm, Go, Java, Scala, Rust, Пифагор, OpenMP |
| 3 | Постановки задач | Асинхронное программирование | БАРС, Haskell, Erlang, JavaScript, Python, C# |

| | | | |
|---|---|---|---|
| 4 | Производитель-ность программ | Высокопроизводитель-ное программирование | Setl, HPF, G, Sparkel, mpC, Sanscript, D, Rest, F# |
| 5 | Ознакомление с па-раллелизмом | Учебное программирование | Logo, Робик, Karel, OZ, A++, СИНХРО, Lsl |

Таким образом, для каждой из трудно решаемых задач параллельных вычислений уже сформирована отдельная удобная парадигма её решения и создан ряд языков программирования, поддерживающих такую парадигму, что подтверждает её статус как парадигмы. Различие между парадигмами проявляется в упорядочении важности средств и методов, используемых при решении отдельных задач, другие задачи требуют иного упорядочения. В каждый момент разработки программы обычно используется одна парадигма. Соответственно и в разных языках программирования выделяется одна ведущая парадигма. Требования к решению достаточно сложных задач параллельных вычислений связано с целым рядом трудных задач, что влечёт необходимость использования разных парадигм на разных этапах их создания и фазах их жизни. При переходе к технологии параллельного программирования важна гарантия получения практичного результата, что требует поддержки полного спектра парадигм, используемых на разных этапах разработки программ. Трудоёмкость использования разных парадигм при решении одной задачи обычно минимизируется созданием много-языковых систем, допускающих по мере необходимости возможность перехода от одной парадигмы к другой без затрат на освоение разных интерфейсов. Это приводит к целесообразности создания мультипарадигмального языка параллельных вычислений, поддерживающего одновременно все основные парадигмы параллелизма и разные уровни абстрагирования – от более низкого, чем принято в языках высокого уровня, до языков сверхвысокого уровня [8].

Как показывает опыт языков БАРС и Haskell, в таких случаях удобно декомпозировать определение ЯП на отдельные подъязыки, поддерживающие основные парадигмы или монады, нацеленные на конкретные модели подготовки и представления программ. Важно на каждом этапе разработки программы локализовать использование одной парадигмы, характеризуемой сравнительно небольшим набором средств и методов в рамках одного способа мышления. Каждая парадигма имеет своё наполнение категорий семантических систем и своё упорядочение их роли в процессе программирования [9, 10].

Средства многопроцессорного программирования обычно опираются на данные и характеристики доступной архитектуры, включая основной многопроцессорный комплекс и взаимосвязи между его элементами. Оперирование комплексом позволяет инициировать процессы функционирования отдельных процессоров, их блокировку, возобновление и отмену процессов. По ходу процессов возможны обмены данными по определённым протоколам, результаты которых можно рассматривать как цель программы. Принятие решений начинается с определения пространства возможных многопроцессорных комплексов, что можно рассматривать как особую разновидность памяти со своей дисциплиной функционирования и взаимодействия элементов. Далее происходит выбор подходящих конфигураций и структур пространства итерирования процессов, предназначенных для выполнения на отдельных процессорах. Затем полученная схема управления процессами наполняется собственно действиями, выполняющими вычисления. Обычно предпочитаются процедуры без рекурсии, освобожденные от ряда сложностей управления вычислениями и побочных эффектов над общей памятью. Строится

многопроцессорная программа, допускающая в динамике реконфигурацию многопроцессорного комплекса[1].

При синхронном (синхронизующем) многопоточном программировании выделяются достаточно чёткие схемы управления вычислениями и разделяются регулярные участки и типичные модели программы, удобные для распараллеливания. Главное отличие от многопроцессорного программирования сводится к абстрагированию от реальных структур конкретных комплексов. Потоки формируются ближе к уровню постановки задачи. Обычно выделяются фрагменты, свободные от побочных эффектов в памяти, и допускается не императивное управление временем исполнения потоков программы с учётом иерархии схемы управления программой и некоторых временных отношений. Принятие решений начинается с выбора стандартных схем управления, используется понятие «пространство итерирования», которое можно структурировать в зависимости от распределения данных и методов их хранения, что может влиять на эффективность и дисциплину обработки многоуровневой памяти. Управление итерированием может использовать произвольные предикаты. Схема управления над пространством итерирования потоков наполняется сравнительно простыми для отладки фрагментами, возможно отлаженными заранее или программируемыми автономно. Для вывода результатов программы выделяются специальные средства формирования результатов вычислений, полученных на равноправных потоках многопоточной программы. Таким образом, получается многопоточная программа с динамически изменяемым пространством потоков над локальной памятью с возможностью эпизодической синхронизации их отдельных фрагментов[2].

Асинхронное программирование нацелено на предельное выражение независимых элементов программы, обусловленных природой решаемой задачи, что может быть базой для максимального распараллеливания при условии представления специальных схем организации вычислений, возможно с учётом специфики доступного оборудования. Начинается принятие решений с выбора общих схем управления действиями и представления условий их срабатывания. Действия могут использовать ту или иную дисциплину обработки памяти. Поддерживается неявное и/или программируемое разнообразие дисциплин доступа к памяти, включая иерархию неоднородный памяти, и схем вычислений – фрагментов, наполняющих общую схему программы процедурами или библиотечными модулями. Возможна схема программируемых синхросетей, асинхронно управляющая выполнением действий в зависимости от условий их готовности к выполнению.

Для высокопроизводительного программирования необходим переход от отдельного прогона программы к учёту перспектив её многократного применения и улучшения. Появляется возможность использовать недогруженные мощности многопроцессорных комплексов выполнением фрагментов программы в расчёте на предстоящие в будущем прогоны с данными, возможно востребованными в дальнейших прецедентах её отладки и применения. Примерно так организуют программы, предназначенные для сверхбыстрого реагирования на опасные события, например, при прогнозировании цунами. Принятие решений начинается со схем и моделей вычислений, возможно, над общей памятью, но с приоритетом локальной памяти. Управление вычислениями учитывает особенности многократного выполнения программы при её отладке и применении, включая возможность наследования результатов между сеансами и сравнения измеримых характеристик производительности версий программы. Получа-

[1] https://mooc.tsu.ru/mooc-openedu/mpi/ «Курс Параллельное программирование с использованием OpenMP и MPI»

[2] https://habr.com/ru/post/121925/ Курс «Основы MPI»

ется ряд улучшаемых версий программы, выбор одной из которых может учитывать особенности текущих условий применения, включая конфигурацию многопроцессорного комплекса и требования к измеримым характеристикам производительности программ.

Учебное программирование призвано компенсировать недостатки общепринятых образовательных ориентиров на безошибочность, единственность, последовательность, императивность и однозначность решений при решении любых задач. Такие ориентиры затрудняют обучение методам программирования вообще, а параллельных вычислений тем более. Учебное программирование может давать старт ко всем необходимым парадигмам параллельных вычислений на базе опыта применения и конструирования игровых программ типа визуализации роботов. Это достаточная причина для мультипарадигмальности учебных языков программирования, обычно поддерживающих некоторые средства представления параллельных вычислений.

Долгоживущие ЯП, как и новые ЯП нашего века, обычно мультипарадигмальны. Есть основания для вывода, что успешная практика параллельного программирования требует мультипарадигмальной поддержки полного спектра парадигм параллельных вычислений. Необходимо допускать их развитие, пополнение и применение по мере необходимости с возможностью перехода к очередной парадигме без изменения общеязыковой и системной обстановки.

Жаргон современного практического программирования использует понятие «язык программирования» как «входной язык или расширенное подмножество ЯП типовой системой программирования (СП), функционирующей на базе определённой конфигурации оборудования». Различие заключается в том, что СП обычно сопровождает реализацию ЯП расширяемым комплектом библиотечных модулей и может не поддерживать отдельные сложности семантики ЯП. В результате происходит сглаживание видимых на практике различий между разными ЯиСП. Кроме того, прямые измерения трудоёмкости программирования и производительности программ почти не отражают зависимости результата от принятых программистом решений и выбора конструкций ЯП. Хотя программируемые решения представляются в терминах ЯП, их влияние растворяется в весьма сложном комплексе, наследующем производительность СП и оборудования с большим доминированием характеристик элементной базы и аппаратуры. Таким образом, существует проблема создания методики, позволяющей выявлять такие зависимости совмещением прямых измерений с результатами экспертных оценок особенностей ЯиСП, возможно, отличающихся от оценок ЯП.

Понятийная сложность решаемых задач ПВ выходит за пределы обычного образования и сами понятия программирования обретают более широкое толкование, отчасти противоречащее привычному интуитивному пониманию. Представление результатов оценки понятийной сложности, позволяющее структурировать пространство таких параметров, рассмотрено в статье [11].

Успех функционального программирования на практике существенно зависит от выбора постановок задач, решения которых представляются системами функций, сравнительно простых, не слишком трудоёмких и удобных при отладке [12, 13]. По степени изученности существенно различаются следующие категории постановок задач, влияющие на выбор методов решения задач и трудоёмкость их программирования:

- новые;
- исследовательские;
- практичные;
- точные.

**Новые** постановки задач характеризуются отсутствием доступного прецедента практичного решения задачи, новизной используемых средств или недостатком опыта исполнителей. Задачи параллельных вычислений поставлены ещё в докомпьютерную эпоху и постановки многих таких задач обладают математической точностью [5]. Тем не менее часть задач параллельного программирования их решений приходится рассматривать как новые из-за стремительного обновления информационных технологий, элементной базы и нерешённых образовательных проблем программирования в целом. Любая проблема, не получившая хорошего решения, остаётся в статусе новой задачи независимо от времени её постановки.

**Исследовательские** постановки задач параллельного программирования в настоящее время следуют тенденции функционального подхода и изучения схем структурирования пространства итерирования процессов, позволяющего оптимизировать обработку памяти. Функциональный подход можно рассматривать как методику сведения решений сложных задач к композициям из планарных проекций, достаточно удобных для понимания, анализа и обработки.

**Практичные** постановки математических задач параллельных вычислений, нацеленные на актуальность и удобство применения, преимущественно используют мощность доступных ИТ для воспроизведения математических моделей, ранее ограниченных низкой эффективностью оборудования, а теперь получивших перспективу стать новой информационной революцией.

**Точные** постановки большинства задач параллельного программирования сложились на базе последовательных алгоритмов и включают в себя испытание возможностей используемых средств, связанных с мерой организованности ранее созданной императивной программы. Некоторые сложности связаны с использованием однопроцессорных конфигураций как исходной модели параллельных многопоточных программ. Не исключено, что для предельного случая удобнее рассматривать двухпроцессорные конфигурации. Появление собственно параллельных алгоритмов встречается не так уж часто, хотя в середине 1990-ых годов проводились конференции такого направления.

Обучение параллельным вычислениям входит в образовательные программы многих университетов, что достаточно для понимания их сложности и постановки задач их исследования, нередко на младших курсах и сразу вслед за функциональным программированием до императивного. Проблемой является переход от теории параллельных вычислений к практике реализации высокопроизводительных программ, удовлетворяющих особо сложным, трудно удостоверяемым критериям надёжности и безопасности. Разница в требованиях, нацеленных на понимание смысла решаемых задач, развитие области приложения решений, определение границы применимости полученных результатов и достижение системности при обобщении созданных инструментальных средств важных для решения задач современных ИТ.

Характерной чертой системного подхода как ведущего метода программирования является переход к классам задач при содержательном анализе постановок задач. Границы класса устанавливаются выбором процесса решения задач. Переход к экспериментам на суперкомпьютерах показал, что именно системные решения могут дать весомый вклад в производительность параллельных вычислений, причём такой вклад может превышать теоретические прогнозы. Это можно рассматривать как обоснование необходимости более фундаментального подхода к программированию, особенно к системному программированию и его математическим основам, дающим путь к созданию высокопроизводительных программ. При создании, формировании и исследовании математических моделей как фундаментального базиса для решения особо трудных проблем эффективности, надёжности и безопасности программного обеспечения важную роль играет развитие моделей, связанных со временем и ре-

сурсами, слабо представленных в курсах по классической математике [14]. Строго говоря, программирование, в отличие от математики, вообще не имеет дела ни со значениями, ни с функциями. Программирование работает с данными, которые могут представлять значения функций.

Квалификация системного программиста включает в себя как формирование способностей самостоятельно изобретать решения новых задач, так и навыки ответственно повышать качество готовых решений, наряду с глубоким владением неклассической и фундаментальной математикой и ознакомлением с современной физикой, психологией и лингвистикой, что слишком неудобно для образовательной системы, нацеленной на формальный контроль усвоения устойчивых стереотипов, представленных в учебно-методической литературе и противоречит учебно-методическим традициям.

Экстенсивное развитие ИТ заметно опережает способности человека оперативно осваивать новые возможности аппаратуры и системных средств ИТ, выходящие за пределы пользовательского уровня, поддерживаемого поставщиками инструментария и программных продуктов. Миссия системного программирования – создание инструментария, позволяющего повышать качество информационных систем, включая поиск новых решений по обеспечению надежности и безопасности информационных технологий [5, 15].

**2. Принципы функционального программирования.** Обычно ФП подразумевает поддержку ряда семантических и прагматических принципов, способствующих созданию функциональных моделей на этапе исследовательского компьютерного эксперимента, естественного при решении новых и сложных задач. В их числе ФП поддерживает *внешние* семантические принципы представления алгоритмов, такие, как универсальность, самоприменимость, равноправие параметров, и системно реализационные *внутренние* прагматические принципы поддержки информационной обработки, такие, как гибкость ограничений, неизменяемость данных, строгость результата. Семантическим принципам программист следует при подготовке программы. Прагматические принципы обеспечивает система программирования, освобождая программиста от технически сложных непринципиальных решений, не зависящих от природы задачи. Многие из этих принципов были заложены Дж. Маккарти в первых реализациях языка Lisp [1].

*Универсальность или всеобщность.* Понятия «функция» и «значение» представляются теми же средствами, как и любые данные для компьютерной обработки. Исторически близкое понятие – «принцип хранимой программы».

Этот принцип позволяет строить представления функций из их частей и вычислять части по мере поступления и обработки данных. Нет принципиальных ограничений на манипулирование средствами языка, функциями из определения семантики языка, конструкциями реализации языка в системе программирования и выражениями в программе. Всё, что понадобилось при реализации языка программирования, может пригодиться при его применении. Поскольку не существует препятствий для обработки представлений функций теми же средствами, какими обрабатываются данные, постольку представления функций можно строить из их частей – символов. Их даже можно формировать по ходу процесса вычислений, поступления и обработки информации о них.

*Самоприменимость.* Представления функций прямо или косвенно могут использовать сами себя, что позволяет строить ясные лаконичные рекурсивные символьные формы.

Примеры самоприменимости дают многие математические функции, особенно рекурсивные, такие, как факториал, числа Фибоначчи, суммирование рядов и многие другие, определение которых использует математическую индукцию. В технологии программирования некоторым сходством обладают методы пошаговой или непрерывной разработки программ (рас-

крутки), а также экстремального программирования. Эти методы сводят организацию процесса программирования к ряду шагов, каждый из которых даёт или работоспособную часть программы, или инструмент для выполнения очередных шагов раскрутки.

*Равноправие параметров.* Порядок и метод вычисления параметров не имеют значения. Можно обратить внимание, что параметры при вызове функции вычисляются на одном и том же уровне иерархии, в одном и том же контексте. Поэтому порядок вычисления параметров не имеет значения, может быть произвольным. Все параметры функции определены в контексте, не изменяемом при их вычислении, они не должны быть взаимосвязаны, не могут воздействовать друг на друга. Часть параметров вычисляются до вызова функции, другие могут быть вычислены позднее, но в том же самом контексте. Поэтому представление любой выделенной формулы из определения функции можно представлять и как параметр этой функции. Любая символьная форма в определении функции может быть выделена из него как параметр и, наоборот, подставлена в него. Функции-переменные допустимы равноправно с обычными функциями-константами и могут быть значениями аргументов или выработаны в качестве результатов других функций.

Обеспечение многократного использования данных выполняется с помощью именования. Параметры имеют имена, часто называемые переменными, хотя в функциональном программировании они не меняют значений в рамках одного контекста. Чисто на уровне понятий переменная – именованная часть памяти, предназначенная для многократного доступа к изменяющимся данным, а константа – к неизменным данным. Изменение связи между именем и значением в функциональном программировании возможно лишь при переходе в другой контекст, что эквивалентно изменению имени.

*Прагматические принципы* поддерживает система программирования, точнее, её разработчики. Если семантические принципы достаточно провозгласить в определении языка или парадигмы, то прагматические принципы влекут заметные трудозатраты на уровне создания системы программирования.

*Гибкость ограничений.* Оперативный пересмотр распределения памяти или её освобождения поддержан для профилактики необоснованных простоев памяти.

Гибкость ограничений на пространственные характеристики позволяет исключать необоснованные простои памяти. Бывает, что не хватает памяти принципиально не на всю задачу, а лишь на отдельные блоки данных, возможно, малосущественные для ее решения. Такая проблема в системах функционального программирования решается принципом гибкости ограничений на пространственные характеристики. Эту проблему решает специальная функция – «мусорщик» (garbage collector), пытающаяся при недостатке любой области памяти автоматизировать перераспределение или освобождение памяти. Новые реализации механизма «мусорщик» рационально учитывают преимущества восходящих процессов на больших объемах памяти.

*Неизменяемость данных***.** Представление каждого результата применения функции размещается в новой части свободной памяти без искажения аргументов этой функции, которые могут быть полезны для других функций.

Таким образом, существенно упрощается отладка программ и обеспечивается повторный прогон любых действий. Можно быть уверенным, что все промежуточные результаты сохранены, их можно анализировать и снова использовать в любой момент. Если определение функции – это статическая конструкция, то процесс можно рассматривать как композицию из функций, разворачиваемую по этой конструкции в динамике.

Отдельный, не вполне очевидный, аспект связан с переходом от целых чисел к вещественным, возможно, допускающим изменение точности представления по ходу вычислений. Логически они остаются константами, а система обрабатывает их как переменные.

*Строгость результата***.** Любое число результатов функции может быть представлено как одна символьная форма или структура данных или специальный код, из которых при необходимости можно выбрать нужный результат.

Нередко этот принцип трактуется как требование однозначности функций, что приводит к сомнениям в правомерности функций целочисленного деления, извлечения корня, обратных тригонометрических функций и многих других категорий математических функций. Возражения такому пониманию функций можно видеть ещё в первом издании Фихтенгольца [16].

Представление алгоритмов в виде функциональных программ имеет *практически значимые следствия.* Конструктивность, верификация и факторизация вытекают из семантических принципов. Из прагматических принципов следуют скрытые модели непрерывности процессов, обратимости действий и унарных функций, дающие основу для интуитивного выстраивания функциональных моделей, допускающих удобный компьютерный эксперимент. Кроме того, из ряда семантических и прагматических принципов вытекает поддержка подготовки функциональных моделей программ организации параллельных вычислений, сводимых к комплексам недетерминированных потоков.

*Метапрограммирование* является следствием принципа универсальности, позволяющего представления программ обрабатывать так же, как любые данные. Это даёт поддержку мета-компиляции, включая синтаксически управляемые методы генерации и анализа программ, а также однородно-гомогенные представления программ, внешне сохраняющих аналогию или подобие обрабатываемым данным или прототипам, в том числе смешанные и частичные вычисления, оптимизирующие преобразования, макрогенерацию и многое другое, необходимое для создания операционных систем и систем программирования.

*Верификация* основана на связи принципа самоприменения с методами рекурсии, математической индукции, дедуктивного вывода и логики.

Большинство систем верификации программного обеспечения создаются в рамках функционального программирования. Такие системы делают возможным логически выводить отдельные свойства программ и, благодаря этому, обнаруживать некоторые тонкие ошибки. Если представление объекта имеет подобие некоторой логике вывода, то его свойства могут быть выведены с использованием этой логики. В результате увеличиваются надежность и безопасность программ, хотя это и не позволяет решить проблему корректности в полной мере. Трудности связаны с недостаточностью классической логики по отношению к неклассическим логикам программирования, а также с некоторыми интуитивными ошибками при оценке истинности логических и вероятностных формул.

*Факторизация* непосредственно вытекает из принципа равноправия параметров. Любой помеченный фрагмент программы может быть вынесен из её представления и ассоциирован с определённым именем, допуская возможность восстановления исходного представления. Это позволяет декомпозировать программу на автономно развиваемые модули и накапливать правильность программ не только на уровне библиотечных функций, но и на уровне произвольных фрагментов и схем, а также представлять параллельные потоки, ленивые (отложенные) или опережающие вычисления. Можно сказать, что программа приводится в факторизованную форму по тем или иным параметрам в зависимости от цели её преобразования [17]. Благодаря обратимости действий, вытекающей из прагматического принципа неизменяемости данных, процесс отладки обретает сходимость.

Следствия принципов системно-реализационной прагматической поддержки информационной обработки в системах функционального программирования выражаются в использовании интуитивных моделей, таких, как непрерывность процессов (бесконечность), обратимость действий и унарные функции.

*Непрерывность процессов* интуитивно следует из прагматической поддержки принципа гибкости ограничений.

После выполнения любой функции можно выполнить ещё одну другую функции (команда STOP не является функцией, у нее нет ни аргументов, ни результатов, это просто сигнал процессору перестать работать). Одновременно с выполнением любой функции можно выполнять другие функции, также и перед выполнением функции можно выполнять другие функции. Это позволяет значительную часть работы программы основывать на модели неограниченной памяти без особого беспокойства о ее границах и разнообразии характеристик скорости доступа к различным структурам данных. Во многих языках функционального программирования поддерживается имитация работы с бесконечными структурами данных [9, 18].

*Обратимость действий* базируется на иллюзии неизменяемости данных, механизмы которой скрыты в СП, их применение почти не требует заботы при подготовке программы и основного объёма отладки. Это позволяет поддерживать механизм мемоизации[3] функций на ранее обработанных аргументах. Фактически необходимые изменения данных, такие, как повторное использование памяти, просто автоматизированы. Программист может позволить себе не вмешиваться в реализацию таких средств до тех пор, пока не возникнут проблемы с производительностью.

*Унарные функции* естественно выводятся из принципа строгого результата, позволяющего при необходимости любое число представленных результатов рассматривать как общую структуру из них. Поскольку результаты часто являются аргументами объемлющих функций, логично возникает и сопутствующий принцип унарных функций. Функция любого числа аргументов может быть преобразована в функцию одного аргумента.

**3. Приложения для параллельных вычислений.** Параллелизм опирается на целый комплекс семантических и прагматических принципов функционального программирования, что позволяет при необходимости рассматривать любой объем представленных данных и при необходимости реорганизовывать пространство потоков. Такой комплекс делает функциональное программирование удобным для работы с программами, направленными на организацию независимых параллельных процессов. Прежде всего, это принцип равноправия параметров, который гарантирует одинаковый контекст при вычислении параметров функции. Становится возможным представлять независимые потоки и объединять их в многопоточные или многопроцессорные программы, в общий программный комплекс. Кроме того, в параллелизме используются принципы строгого результата и универсальности, что позволяет при необходимости учитывать любое количество представленных потоков и реорганизовывать пространство потоков. Их конкретизация сводится на уровне семантики к методам оттеснения мало вероятных вычислений, балансировки нагрузки, выстраивания пространства итераций.

Чистое функциональное программирование не совсем удобно для моделирования взаимодействующих и императивно синхронизированных процессов. Для организации таких процессов требуется дополнительное уточнение принципов, упрощающее подготовку высокопроизводительных программ параллельных вычислений.

*Оттеснение мало вероятных* вычислений направлено на предотвращение затрат на формирование и выполнение чрезмерного количества потоков, соответствующих слишком редким ситуациям.

Принцип универсальности имеет два аспекта – равноправие программ с данными и полноту определения функции. При решении задач параллельных вычислений сохраняется аспект равноправия программ с данными, традиционно востребованный в задачах операционных систем и управления процессами. Полнота функции, удобная для построения

[3] Мемоизоляция – сохранение результатов выполнения функций для предотвращения повторных вычислений функций.

программ из универсально отлаженных модулей, может создавать проблемы в многопроцессорных программах из-за увеличения количества потоков, связанных с различными редкими диагностическими ситуациями.

Любой фрагмент, выполнение которого маловероятно или невозможно, может быть представлен как отложенное действие. Ветви для практически неактуальных ситуаций можно перенести в отладочную версию. Иногда эту проблему преодолевают путем выбора выражений, не требующих ветвления, чаще – путем проверки типов данных. Объем необходимой диагностики можно частично сократить за счет статического анализа типов данных.

*Балансировка нагрузки* сокращает реальное время выполнения многопоточной программы, позволяет выравнивать нагрузку процессоров и избегать их необоснованных простоев, что можно рассматривать как распространение принципа гибкости ограничений на временные границы.

Ленивые или опережающие вычисления дают возможность быстро и оперативно перераспределять нагрузку. Сложное определение функции иногда можно свести к двум функциям, первая из которых выполняет часть определения, откладывая выполнение остальных, а вторая возобновляет выполнение отложенных частей. Возможно, выполнение второй функции произойдет одновременно со второй или даже раньше. В языке mpC предлагается объем вычислений оперативно перераспределять при обнаружении неравномерной загрузки процессоров [19].

Само-применение в виде рекурсивных функций часто рассматривается как осложнение, приводящее к опасному росту стека. Здесь следует отметить, что многие системы функционального программирования предлагают ряд практичных решений. Кроме того, в функциональном программировании существуют отложенные действия, мемоизация, восходящая рекурсия, методы динамического программирования и оптимизация рекурсий за счет сведения к циклам, что во многих случаях позволяет практически устранить чрезмерное разбухание стека. Поддержку работы со стеком в рамках принципа гибкости ограничений можно поддерживать более эффективно, чем в большинстве языков и систем программирования. Факторизация программ на схемы и фрагменты позволяет разделять компоненты по уровню сложности отладки и наследовать корректность ранее отлаженных составляющих. Используются функции, не требующие предварительного вычисления параметров, например макротехника.

*Пространство итераций* используется в качестве основного параметра для управления распараллеливанием циклов.

Любое конечное множество может работать, как пространство итераций для определенной на нем функции. Если есть набор данных, на котором вычисление функции на одном элементе не требует ее результатов на других, то использование такого набора в качестве пространства итераций удобно для одновременного выполнения этой функции на всех элементах набора. Подобная техника распространения операций и функций имеется в языках APL, Альфа, БАРС и Sisal. Роль равноправия параметров растет, она обеспечивает решение проблем реорганизации потоков при настройке на различные конфигурации многопроцессорных систем, требующих декомпозиции программы на схемы и фрагменты. Многие работы в сфере суперкомпьютерных вычислений по существу являются поиском удобного пространства итераций. Техника и концепция пространств итераций убедительно поддерживаются в языке Sisal, в котором пространства итераций строятся над перечислимыми множествами с использованием операций скалярного и декартового произведения.

Несколько сложнее с прагматическими принципами, требующими пересмотра системных решений на уровне разработки СП. Здесь важную роль играют автоматическое

распараллеливание, идентичность повторных прогонов и представление много-контактных фрагментов.

*Автоматическое распараллеливание* заключается в извлечении из программы автономных частей, допускающих независимое выполнение. Если существует функция с известным временем выполнения, которую можно разложить на ряд независимых функций с меньшим временем счёта, то они могут быть выполнены в произвольном порядке, и время выполнения исходной функции, как правило, будет строго меньше.

*Идентичность повторных прогонов* для целей отладки программ и измерения их производительности. Переход к суперкомпьютерам показал, что при слишком большом числе процессоров исчезает нужная для отладки и измерений возможность пронаблюдать повторное исполнение программы. При очередном прогоне могут быть сбои на других процессорах. Функциональное программирование здесь может допускать специальные интерпретации программы, учитывающие протоколы и результаты ранее выполненных прогонов с отслеживанием идентичности исполнения. Обеспечение идентичности прогонов требует дальнейшего развития средств отладки программ.

*Многоконтактные фрагменты*, такие, как схемы управления или операции, имеющие несколько операндов и дающие более одного результата, на первый взгляд, противоречат принципу строгого результата. Однако возможность перехода от строгого результата позволяет строить многопоточные функции, которые принимают параметры от ряда одноранговых потоков и генерируют ряд результатов в терминах числа потоков. Таким образом, можно, как и в функциональном языке параллельного программирования Sisal, переключиться на операции, которые отображают один ряд операндов в другой ряд результатов. Это может соответствовать структуре некоторых особо эффективных аппаратных узлов и, таким образом, позволяет представлять более эффективные решения [20].

**4. Повышение производительности.** При переходе к производству многократно применяемых программ и реализации особо важных параллельных вычислений резко меняются предпочтения. ***Успех приложения и производительность программ становятся важнее, чем их формальная корректность и эффективность***. Чистое функциональное программирование можно рассматривать как метод функционального моделирования при создании прототипов программ для решения сложных проблем. Более широкая парадигма производственного функционального программирования позволяет переходить от таких функциональных моделей и прототипов к более эффективным структурам данных, наследуя ранее достигнутую правильность и принимая практичные решения и компромиссы по обработке данных в зависимости от реальных условий, если это действительно необходимо.

В дополнение к принципам и следствиям в реальных языках программирования и системах производственное функциональное программирование обычно включает компромиссные механизмы, внешне в языке программирования выглядящие как специальные функции. Например, Lisp 1.5, Clisp, Cmucl и другие члены семейства Lisp обычно предоставляют следующие функции практичного компромисса:

- **контроль типов данных** смягчает принцип *универсальности* функциями статического и динамического анализа типов данных и рангов функций;
- **схемы циклов**, моделирующие несколько расширенное множество знакомых механизмов управления вычислениями, позволяют преодолеть типичные опасения по поводу сложности реализации принципа *само-применения* на базе рекурсии;
- **возможность восстановления данных** при использовании деструктивных функций сопровождается безопасными аналогами, позволяющими исключить чрезмерное потребление памяти, частично противодействуя принципу *неизменности данных*;

- **программируемые прогнозы** объема памяти и скорости выполнения позволяют, по мере необходимости, императивно выполнять в программе распределение памяти для нейтрализации грубоватых механизмов принципа *гибкости ограничений*;
- **псевдо-функции**, помимо генерации результата функции, выполняют нагрузку в виде воздействия на внешнюю память или взаимодействия с устройствами, включая операции ввода-вывода и файловые операции, что несколько влияет на принципы *равноправия параметров и неизменяемости данных*;
- **мемоизация** позволяет радикально снижать сложность многократно повторяемых вычислений, делая понятную уступку успешному опыту против принципов *строгого результата и верификации* [21].

Общее взаимодействие принципов, следствий, приложений и практичных компромиссов в системах функционального программирования представлено в табл. 2.

**Таблица 2.** Ключевые взаимосвязи принципов и механизмов ФП

| | Семантика | Прагматика |
|---|---|---|
| Принципы | универсальность,<br>само-применение,<br>равноправие параметров | гибкость ограничений,<br>неизменяемость данных,<br>строгость результата |
| Следствия | мета-программирование,<br>верификация,<br>факторизация | непрерывность процессов,<br>обратимость действий,<br>унарные функции |
| Приложения к параллельным вычислениям | оттеснение мало вероятных вычислений,<br>балансировка нагрузки,<br>пространства итераций | автоматическое распараллеливание,<br>идентичность повторных прогонов,<br>многоконтактные фрагменты |
| Практичный компромисс | контроль типа данных,<br>схемы циклов,<br>восстановление данных | программируемый учет прогнозов,<br>псевдо-функции доступа к устройствам,<br>мемоизация |

В рамках функционального программирования возможен подход, позволяющий учитывать особенности решаемых задач и парадигм программирования, необходимых для решения проблем параллельных вычислений, влияющих на выбор методов их решения, в зависимости от приоритетов в выборе языковых средств и реализационных конструкций. Можно наметить линию, позволяющую сравнивать языки, выделяя сопоставимые примеры программ, и особенности базовых средств и реализационных решений в системах программирования, нацеленных на улучшения создаваемых программных продуктов. Программирование давно уже стало массовой профессией, требующей объективных метрик для оценки качества программируемых решений. Парадигмальные ошибки, обнаруживаемые при эксплуатации привычных систем программирования на современном многопроцессорном и сетевом оборудовании, показывают, что часть из них были просто незаметны до появления сетей, мобильных устройств и суперкомпьютеров.

В целом следует отметить, что следствия из семантических и прагматических принципов функционального программирования и высокая моделирующая сила аппарата функций, расширенные специальными функциями практичных компромиссов, позволяют полезно дополнять основные парадигмы параллельных вычислений и поддерживать работы по повышению производительности программ. В этом плане существенный вклад дают мемоизация, отложенные и опережающие вычисления, возможность синтаксического

конструирования и декомпозиции программ на автономно развиваемые модули, а на более общем уровне – метапрограммирование, позволяющее строить специализированные проблемно ориентированные DSL-языки. Синтаксическое конструирование – механизм надёжного использования прототипов и структур обрабатываемых данных. Декомпозиция программ приводит к выводу автономно развиваемых модулей, изменение которых не требует повторного программирования смежных модулей. Метапрограммирование может быть применено для создания подъязыков, учитывающих индивидуальные особенности процессоров, а также для обустройства переходов от одного языка к другому.

**Заключение.** В феврале 2021 года состоялась 22-я конференция, посвященная современным тенденциям функционального программирования [7]. Представленные доклады убедительно показали нацеленность функционального программирования на решение основных проблем организации параллельных вычислений.

Многие вопросы пока не получили практичного ответа. Появление новых парадигм можно связать с проявлением круга новых задач, решение которых вызывает трудности в рамках привычных парадигм. Не ясно, насколько целесообразно взаимодействие парадигм. Остаются в стороне образовательные проблемы овладения новыми парадигмами. Кроме того, особенности парадигм лишь отчасти выражаются на уровне представления программы, часть являются требованиями к прагматике системно-реализационной поддержки в ЯиСП. Граница между семантикой программируемых решений и прагматикой их системной поддержки у каждой парадигмы своя.

Похоже, путь к решению проблем параллельного программирования лежит через создание многоуровневого мультипарадигмального языка для поддержки основных парадигм параллельного программирования, включая средства низкого и сверхвысокого уровней, дополненных метапарадигмой функционального программирования. Собственно функциональное программирование выполняет роль проектно-конструкторского бюро для производственного программирования в рамках любой парадигмы и любого языка программирования.

## СПИСОК ИСТОЧНИКОВ

1. McCarthy J. LISP 1.5 Programming Mannual. The MIT Press, Cambridge, 1963, 106 p.
2. Backus J. Can programming be liberated from the von Neumann style? A functional stile and its algebra of programs. Commun. ACM 21, 8, 1978, pp. 613-641.
3. Gorodnyaya L. Method of paradigmatic analysis of programming languages and systems. CEUR Workshop Proceedings 2543 (2020), pp. 149-158.
4. Gorodnyaya L. On the presentation of the results of the analysis of programming languages and systems. CEUR Workshop Proceedings 2260 (2018), pp. 152–166.
5. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления / В.В., Воеводин, Вл.В Воеводин. – СПб.: БХВ-Петербург, 2002. – 608 с.
6. Городняя Л.В. О неявной мультипарадигмальности параллельного программирования / Л.В. Городняя //Научный сервис в сети Интернет: труды XXIII Всероссийской научной конференции (20-23 сентября 2021 г., онлайн). – М.: ИПМ им. М.В.Келдыша, 2021. – С.104-116.
7. Pieter Koopman, Steffen Michels, and Rinus Plasmeijer Dynamic Editors for Well-Typed Expressions. Trends in Functional programming/ 22nd internationa Symposium, TFP 2021, February 17-19, 2021 Springer, LNCS 12834, pp. 44-66.

8. Schwartz Jacob T. "Set Theory as a Language for Program Specification and Programming". Courant Institute of Mathematical Sciences, New York University, 1970[4].
9. Душкин Р. Функциональное программирование на языке Haskel / Р. Душкин. – ДМК-Пресс, 2016. – 606 с. – URL: https://www.labirint.ru/books/497084/
10. Котов В.Е. МАРС: архитектуры и языки для реализации параллелизма / Котов В.Е. // Системная информатика. Вып 1. Проблемы современного программирования. – Новосибирск: Наука. Сиб. отд-ние, 1991. – С.174-194.
11. Городняя Л.В. Систематизации парадигм программирования по приоритетам принятия решений / Л.В. Городняя // Электронные библиотеки, 2020. – Т. 23 ,ч. 2. – № 4. – С. 666-696. – DOI:10.26907/1562-5419-2020-23-4-666-696
12. Лавров С.С. Функциональное программирование /С.С. Лавров// Компьютерные инструменты в образовании, 2002. – №2-4.
13. Лавров С.С., Городняя Л.В. Функциональное программирование. Интерпретатор языка Лисп / С.С.Лавров, Л.В. Городняя //Компьютерные инструменты в образовании. – С-Пб, 2002. – № 5.
14. Мальцев А.И. Алгоритмы и рекурсивные функции / Мальцев А.И. – М.: Наука, 1965 г. – 392 с.
15. Бурдонов И.Б., Косачев. А.С. Семантики взаимодействия с отказами, дивергенцией и разрушением. Часть 2. Условия конечного полного тестирования / И.Б. Бурдонов, А.С. Косачев //Вестник Томского Государственного Университета, 2011. – № 2(15).
16. Фихтенгольц Г.М. Основы математического анализа. Том 1 / Г.М. Фихтенгольц. – М: «Наука».,1968. – 440 с.
17. Городняя Л.В., Кирпотина И.А. Методика декомпозиции сложных информационных систем / Л.В. Городняя, И.А Кирпотина И.А // 30-я Международная конференция "Информационные технологии в образовании 2019" (ИТО-Троицк-Москва-2019), 2019 г.
18. Сошников Д. В. Функциональное программирование на языке F# / Д. В. Сошников. –М.: ДМК Пресс, 2011.
19. Ластовецкий А.Л. Предварительное сообщение о языке программирования mpC / А.Л. Ластовецкий // Вопросы кибернетики: Приложения системного программирования. Научный совет по комплексной проблеме "Кибернетика" РАН. Москва, 1995. – С. 20-39.
20. Касьянов В.Н., Гордеев Д.С., Золотухин Т.А. Система облачного параллельного программирования CPPS: визуализация и верификация Cloud Sisal программ / В.Н. Касьянов, Д.С. Гордеев, Т.А. Золотухин // Конструирование и оптимизация программ. – Новосибирск: НГУ, 2020.
21. Файфель Б.Л. Мемоизация в Лиспе / Б.Л. Файфель. – URL: https://habr.com/ru/post/576278/

***Городняя Лидия Васильевна,*** *к.ф.-м.н., доцент, с.н.с., Институт систем информатики СО РАН, gorod@iis.nsk.su, Россия, г. Новосибирск, проспект Академика Лаврентьева, 6*

[4] Данфорд Н., Шварц Дж. Линейные операторы. Том 2. Спектральная теория. Самосопряженные операторы в гильбертовом пространстве. М.: Мир, 1966

UDC 004.43 (042.4)
doi:10.38028/ESI.2022.25.1.009

# On functional programming in the organization of parallel computing

**Lidia V. Gorodnyaya**
A.P. Ershov Institute of Informatics Systems, Siberian Branch of the Russian Academy of Sciences, Russia, Novosibirsk, *lidvas@gmail.com*

**Abstract.** The article is devoted to the results of the analysis of modern trends in functional programming, considered as a meta-paradigm for solving the problems of organizing parallel computing and multi-threaded programs for multi-processor complexes and distributed systems. Taking into account the multi-paradigm nature of parallel programming languages, here we used the results of paradigm analysis, which made it possible to find out a number of features and predict the course of the processes of using programs, as well as their study and development. There is reason to believe that functional programming can improve the performance of programs by preparing their prototypes in advance. A description of the semantic (universality, self-applicability, equality of parameters) and pragmatic (flexibility of constraints, immutability of data, rigor of the result) principles of functional programming is given, as well as the consequences of these principles aimed at reducing labor intensity and improving the technology of prototyping and debugging programs (constructiveness, verification, factorization, continuity of processes, reversibility of actions). The role of paradigmatic decomposition of programs in the technology of developing long-lived programs, which are often parallel computation programs, is noted. The perspective of functional programming as an auxiliary universal meta-paradigm for solving complex and important problems, burdened with difficult to verify and poorly compatible requirements of efficiency, reliability, safety and improvement opportunity, is especially emphasized. A variety of paradigmatic characteristics inherent in the preparation and debugging of long-lived parallel computing programs are shown.

**Keywords:** functional programming, parallel computing, programming languages, programming paradigms, multiparadigm

REFERENCES

1. McCarthy J. LISP 1.5 Programming Mannual. The MIT Press, Cambridge, 1963, 106 p.
2. Backus J. Can programming be liberated from the von Neumann style? A functional stile and its algebra of programs. Commun. ACM 21, 8, 1978, pp. 613-641.
3. Gorodnyaya L. Method of paradigmatic analysis of programming languages and systems. CEUR Workshop Proceedings 2543 (2020), pp. 149-158.
4. Gorodnyaya L. On the presentation of the results of the analysis of programming languages and systems. CEUR Workshop Proceedings 2260 (2018), pp. 152–166.
5. Voyevodin V.V., Voyevodin Vl.V. Parallel'nyye vychisleniya [Parallel computing] / V.V. Voyevodin, Vl.V. Voyevodin – SPb.: BKHV-Peterburg = St. Petersburg: BHV-Petersburg, 2002. – 608 p.
6. Gorodnyaya L.V. O neyavnoy mul'tiparadigmal'nosti parallel'nogo programmirovaniya [On the implicit multi-paradigm of parallel programming] / L.V. Gorodnyaya // Nauchnyy servis v seti Internet: trudy XXIII Vserossiyskoy nauchnoy konferentsii (20-23 sentyabrya 2021 g., onlayn) = Scientific service on the Internet: Proceedings of the XXIII All-Russian Scientific

Conference (September 20-23, 2021, online). – Moscow: IPM im. M.V. Keldysh, 2021, pp.104-116 p.

7. Pieter Koopman, Steffen Michels, and Rinus Plasmeijer Dynamic Editors for Well-Typed Expressions. Trends in Functional programming/ 22nd internationa Symposium, TFP 2021, February 17-19, 2021/ Springer, LNCS 12834, pp. 44-66.
8. Schwartz Jacob T. "Set Theory as a Language for Program Specification and Programming". Courant Institute of Mathematical Sciences, New York University, 1970[5].
9. Dushkin R. Funktsional'noye programmirovaniye na yazyke Haske [Functional programming in Haskel] / R. Dushkin. – DMK-Press, 2016, – 606 p.
10. Kotov V.Ye. MARS: arkhitektury i yazyki dlya realizatsii parallelizma [MARS: architectures and languages for implementing parallelism] / V.Ye. Kotov // Sistemnaya informatika. Vyp 1. Problemy sovremennogo programmirovaniya = System informatics. Issue 1. Problems of modern programming. – Novosibirsk: Nauka, Sib. otd-niye= Science. Sib. Dept, 1991, – pp.174-194.
11. Gorodnyaya L.V. Sistematizatsii paradigm programmirovaniya po prioritetam prinyatiya resheniy [Systematization of programming paradigms according to decision-making priorities] / L.V. Gorodnyaya /Elektronnyye biblioteki = Digital Libraries, Volume 23, № 4 (2020), Part 2, – Pp. 666-696. DOI:10.26907/1562-5419-2020-23-4-666-696.
12. Lavrov S.S. Funktsional'noye programmirovaniye [Functional programming] / S.S. Lavrov // Komp'yuternyye instrumenty v obrazovanii = Computer tools in education, 2002, № 2-4.
13. Lavrov S.S., Gorodnyaya L.V. Funktsional'noye programmirovaniye. Interpretator yazyka Lisp [Functional programming. Lisp language interpreter] / S.S. Lavrov, L.V. Gorodnyaya // Komp'yuternyye instrumenty v obrazovanii = Computer tools in education, – S-Pb, 2002, № 5.
14. Mal'tsev A.I. Algoritmy i rekursivnyye funktsii [Algorithms and recursive functions] / A.I. Mal'tsev. – M.: Nauka = Science, 1965, 392 p.
15. Burdonov I.B., Kosachev A.S. Semantiki vzaimodeystviya s otkazami, divergentsiyey i razrusheniyem. Chast' 2. Usloviya konechnogo polnogo testirovaniya [Semantics of interaction with failures, divergence and destruction. Part 2. Conditions for final full testing] / I.B. Burdonov, A.S. Kosachev. Vestnik Tomskogo Gosudarstvennogo Universiteta = Bulletin of Tomsk State University, № 2(15), 2011.
16. Fikhtengol'ts G.M. Osnovy matematicheskogo analiza [Fundamentals of mathematical analysis] / G.M. Fikhtengol'ts. – M: «Nauka»= "Science", Tom 1, 1968, – 440 p.
17. Gorodnyaya L.V., Kirpotina I.A. Metodika dekompozitsii slozhnykh informatsionnykh system [Decomposition methodology for complex information systems] / L.V. Gorodnyaya, I.A. I.A. // 30-ya Mezhdunarodnaya konferentsiya "Informatsionnyye tekhnologii v obrazovanii 2019" (ITO-Troitsk-Moskva-2019). 25 iyunya 2019 g. = 30th International Conference "Information Technologies in Education 2019" (ITO-Troitsk-Moscow-2019), June 25, 2019.
18. Soshnikov D.V. Funktsional'noye programmirovaniye na yazyke F# [Functional programming in F#] / D.V. Soshnikov. – Moscow: DMK Press, 2011.
19. Lastovetskiy A.L. Predvaritel'noye soobshcheniye o yazyke programmirovaniya mpC [Preliminary report on the mpC programming language] / A.L. Lastovetskiy // Voprosy kibernetiki: Prilozheniya sistemnogo programmirovaniya. Nauchnyy sovet po kompleksnoy probleme "Kibernetika" RAN = Issues of Cybernetics: Applications of System Programming.

[5] Dunford N., Schwartz J. Linear Operators. Volume 2. Spectral theory. Self-adjoint operators in a Hilbert space. Moscow: Mir, 1966 *(in Russian).*

Scientific Council on the complex problem "Cybernetics" RAS. – Moscow, 1995, – pp. 20-39.

20. Kas'yanov V.N., Gordeyev D.S., Zolotukhin T.A. Sistema oblachnogo parallel'nogo programmirovaniya CPPS: vizualizatsiya i verifikatsiya Cloud Sisal program [Cloud parallel programming system CPPS: visualization and verification of Cloud Sisal programs] / V.N.Kas'yanov? D.S. Gordeyev, T.A. Zolotukhin // Konstruirovaniye i optimizatsiya program = Design and optimization of programs. Novosibirsk: NGU = – Novosibirsk State University, 2020.
21. Fayfel' Memoizatsiya v Lispe [Memoization in Lisp] / B.L. Fayfel' – URL: https://habr.com/ru/post/576278/

***Lidia V. Gorodnyaya**, candidate of Physical and Mathematical Sciences, Associate Professor, Senior Researcher, A.P. Ershov Institute of Informatics Systems, Siberian Branch of the Russian Academy of Sciences, lidvas@gmail.com**, Russia, Novosibirsk, Akademika Lavrentiev Avenue, 6***

*Статья поступила в редакцию 21.12.2021; одобрена после рецензирования 20.01.2022; принята к публикации 25.01.2022.*

*The article was submitted 12.21.2021; approved after reviewing 01.20.2022; accepted for publication 01.25.2022.*