Параллельное программирование

Параллельные вычисления - способ организации компьютерных вычислений, при котором программы разрабатываются, как набор взаимодействующих вычислительных процессов, работающих асинхронно и при этом одновременно.

Параллельное программирование - это техника программирования, которая использует преимущества многоядерных или многопроцессорных компьютеров и является подмножеством более широкого понятия многопоточности (multithreading).

Существуют различные способы написания программ, (разделение условное) :

- Последовательное программирование с дальнейшим автоматическим распараллеливанием.
- Непосредственное формирование потоков параллельного управления, с учетом особенностей архитектур параллельных вычислительных систем или операционных систем.
- Описание параллелизма без использования явного управления обеспечивается заданием только информационных связей. Предполагается, что программа будет выполняться на вычислительных системах с бесконечными ресурсами, операторы будут запускаться немедленно по готовности их исходных данных.

Достижение параллелизма возможно только при выполнении следующих требований к архитектурным принципам построения вычислительной среды:

независимость функционирования отдельных устройств ЭВМ – данное требование относится в равной степени ко всем основным компонентам вычислительной системы: к устройствам ввода-вывода, обрабатывающим процессорам и устройствам памяти;

избыточность элементов вычислительной системы – организация избыточности может осуществляться в следующих основных формах:

- *использование специализированных устройств*, таких, например, как отдельные процессоры для целочисленной и вещественной арифметики, устройства многоуровневой памяти (регистры, *кэш*);

- *дублирование устройств ЭВМ* путем использования, например, нескольких однотипных обрабатывающих процессоров или нескольких устройств оперативной памяти.

Дополнительной формой обеспечения параллелизма может служить *конвейерная* реализация обрабатывающих устройств, при которой выполнение операций в устройствах представляется в виде исполнения последовательности составляющих операцию подкоманд. Как результат, при вычислениях на таких устройствах на разных стадиях обработки могут находиться одновременно несколько различных элементов данных.

При *конвейерной* реализации параллелизм ограничен числом стадий

- В идеале времена работы каждой стадии должны быть одинаковыми Самая медленная стадия становится узким местом
- Комбинирование и декомпозиция стадий
- Распараллеливание медленной стадии
- Времена заполнения и опустошения конвейера
-

При рассмотрении проблемы организации параллельных вычислений следует различать следующие возможные режимы выполнения независимых частей программы:

- *многозадачный режим* (*режим разделения времени*), при котором для выполнения нескольких процессов используется единственный процессор. Данный режим является псевдопараллельным, когда активным (исполняемым) может быть один, единственный процесс, а все остальные процессы находятся в состоянии ожидания своей очереди. Применение *режима разделения времени* может повысить эффективность организации вычислений (например, если один из процессов не может выполняться из-за ожидания вводимых данных, процессор может быть задействован для выполнения другого, готового к исполнению процесса). Кроме того, в данном режиме проявляются многие эффекты параллельных вычислений (необходимость взаимоисключения и синхронизации процессов и др.), и, как результат, этот режим может быть использован при начальной подготовке параллельных программ;

- параллельное выполнение, когда в один и тот же момент может выполняться несколько команд обработки данных. Такой режим вычислений может быть обеспечен не только при наличии нескольких процессоров, но и при помощи конвейерных и векторных обрабатывающих устройств;
- *распределенные вычисления* - данный термин обычно применяют для указания параллельной обработки данных, при которой используется несколько обрабатывающих устройств, достаточно удаленных друг от друга, в которых передача данных по линиям связи приводит к существенным временным задержкам. Как результат, эффективная обработка данных при таком способе организации вычислений возможна только для параллельных алгоритмов с низкой интенсивностью потоков межпроцессорных передач данных. Перечисленные условия являются характерными, например, при организации вычислений в многомашинных вычислительных комплексах, образуемых объединением нескольких отдельных ЭВМ с помощью каналов связи локальных или глобальных информационных сетей.

Классификация вычислительных систем

Одним из наиболее распространенных способов классификации ЭВМ является систематика Флинна (Flynn, 1966 г.), в рамках которой основное внимание при анализе архитектуры вычислительных систем уделяется способам взаимодействия последовательностей (потоков) выполняемых команд и обрабатываемых данных. При таком подходе различают следующие основные типы систем

Название класса	Описание класса
SISD (single instruction stream / single data stream) или ОКОД (Одиночный поток Команд, Одиночный поток Данных)	Одиночный поток команд и одиночный поток данных (исполнение одним процессором одного потока команд, обрабатывающего данные, хранящиеся в одной памяти). К этому классу относятся, классические последовательные машины, или иначе, машины фон-неймановского типа.
SIMD (single instruction stream / multiple data stream) или ОКМД (одиночный поток команд, множественный поток данных)	Одиночный поток команд и множественный поток данных. В архитектурах подобного рода сохраняется один поток команд, включающий, в отличие от предыдущего класса (SISD), векторные команды, что позволяет выполнять одну арифметическую операцию сразу над многими данными - элементами вектора.
MISD (multiple instruction stream / single data stream) или МКОД (Множественный поток Команд, Одиночный поток Данных)	Множественный поток команд и одиночный поток данных. Определение подразумевает наличие в архитектуре многих процессоров, обрабатывающих один и тот же поток данных.
MIMD (multiple instruction stream / multiple data stream) или МКМД (Множественный поток Команд, Множественный поток Данных)	Множественный поток команд и множественный поток данных. Этот класс предполагает, что в вычислительной системе есть несколько устройств обработки команд, объединенных в единый комплекс и работающих каждое со своим потоком команд и данных.

Мультипроцессоры

Для дальнейшей систематики мультипроцессоров учитывается способ построения общей памяти. Первый возможный вариант – использование единой (централизованной) общей памяти (shared memory) (рис. 1 а). Такой подход обеспечивает однородный доступ к памяти (uniform memory access или UMA) и служит основой для построения векторных параллельных процессоров (parallel vector processor или PVP) и симметричных мультипроцессоров (symmetric multiprocessor или SMP). Среди примеров первой группы - суперкомпьютер Cray T90, ко второй группе относятся IBM eServer, Sun StarFire, HP Superdome, SGI Origin и др.

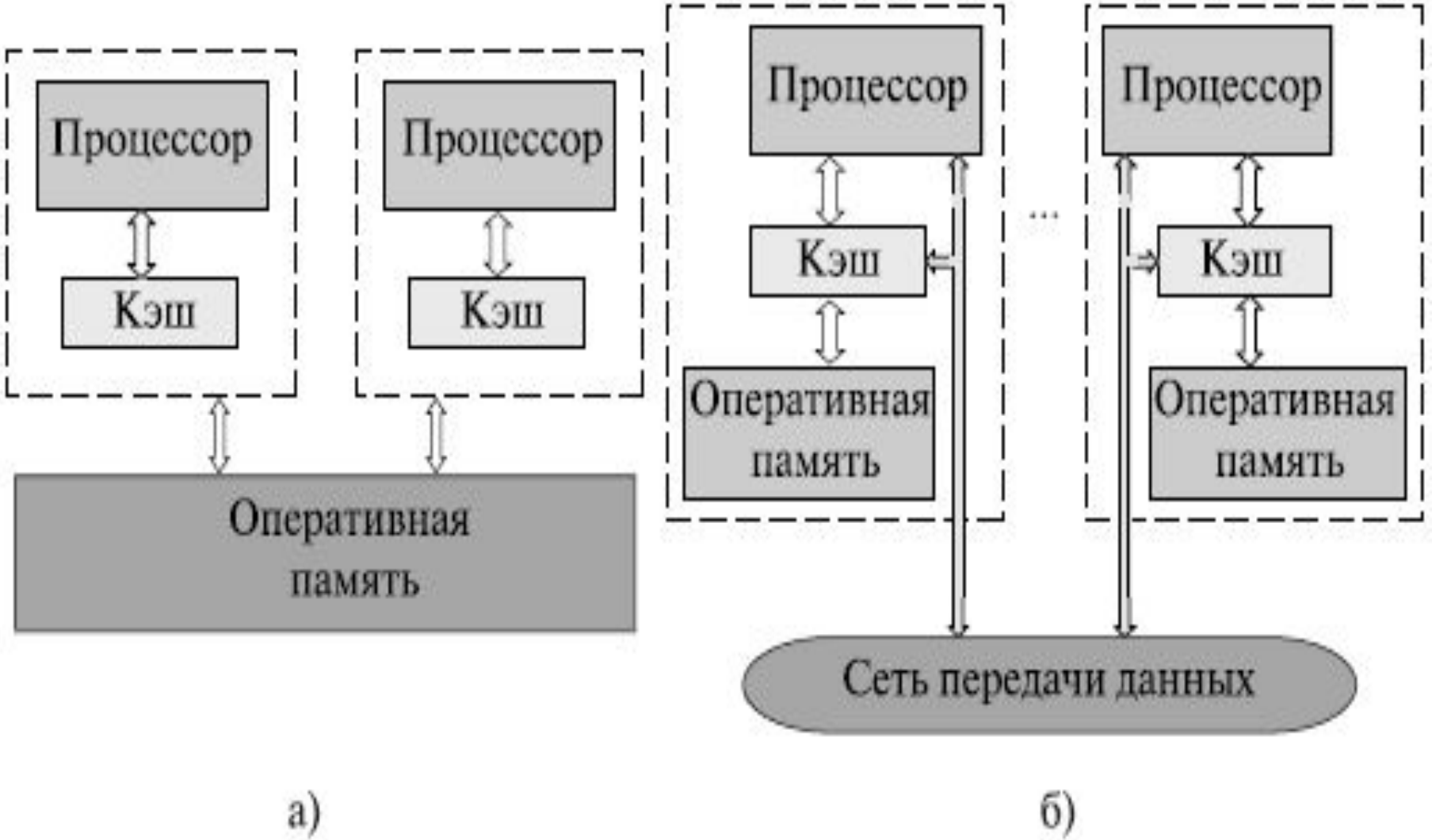


Рис. 1. Архитектура многопроцессорных систем с общей (разделяемой) памятью: системы с однородным (а) и неоднородным (б) доступом к памяти

Одной из основных проблем, которые возникают при организации параллельных вычислений на такого типа системах, является доступ с разных процессоров к общим данным и обеспечение, в связи с этим, однозначности (когерентности) содержимого разных кэшей (cache coherence problem). Дело в том, что при наличии общих данных копии значений одних и тех же переменных могут оказаться в кэшах разных процессоров. Если в такой ситуации (при наличии копий общих данных) один из процессоров выполнит изменение значения разделяемой переменной, то значения копий в кэшах других процессоров окажутся не соответствующими действительности и их использование приведет к некорректности вычислений. Обеспечение однозначности кэшей обычно реализуется на аппаратном уровне – для этого после изменения значения общей переменной все копии этой переменной в кэшах отмечаются как недействительные и последующий доступ к переменной потребует обязательного обращения к основной памяти.

Следует отметить, что необходимость обеспечения когерентности приводит к некоторому снижению скорости вычислений и затрудняет создание систем с достаточно большим количеством процессоров.

Наличие общих данных при параллельных вычислениях приводит к необходимости синхронизации взаимодействия одновременно выполняемых потоков команд. Так, например, если изменение общих данных требует для своего выполнения некоторой последовательности действий, то необходимо обеспечить взаимоисключение (mutual exclusion), чтобы эти изменения в любой момент времени мог выполнять только один командный поток. Задачи взаимоисключения и синхронизации относятся к числу классических проблем, и их рассмотрение при разработке параллельных программ является одним из основных вопросов параллельного программирования.

Общий доступ к данным может быть обеспечен и при физически распределенной памяти (при этом, естественно, длительность доступа уже не будет одинаковой для всех элементов памяти) (см. рис. 1 б). Такой подход именуется неоднородным доступом к памяти (non-uniform memory access или NUMA). Среди систем с таким типом памяти выделяют:

- системы, в которых для представления данных используется только локальная кэш-память имеющихся процессоров (cache-only memory architecture или COMA); примерами являются KSR-1 и DDM;
- системы, в которых обеспечивается когерентность локальных кэшей разных процессоров (cache-coherent NUMA или CC-NUMA); среди таких систем: SGI Origin 2000, Sun HPC 10000, IBM/Sequent NUMA-Q 2000;
- системы, в которых обеспечивается общий доступ к локальной памяти разных процессоров без поддержки на аппаратном уровне когерентности кэша (non-cache coherent NUMA или NCC-NUMA); например, система Cray T3E.

Мультикомпьютеры

Мультикомпьютеры (многопроцессорные системы с к данным, располагаемым на других процессраспределенной памятью) уже не обеспечивают общего доступа ко всей имеющейся в системах памяти (no-remote memory access или NORMA) (см. рис. 2). При всей схожести подобной архитектуры с системами с распределенной общей памятью (рис. 1.б), мультикомпьютеры имеют принципиальное отличие: каждый процессор системы может использовать только свою локальную память, в то время как для доступа к данным, необходимо явно выполнить операции передачи сообщений (message passing operations). Данный подход применяется при построении двух важных типов многопроцессорных вычислительных систем - массивно-параллельных систем (massively parallel processor или MPP) и кластеров (clusters).

Рис. 2. Архитектура многопроцессорных систем с распределенной памятью

Под кластером обычно понимается множество отдельных компьютеров, объединенных в сеть, для которых при помощи специальных аппаратно-программных средств обеспечивается возможность унифицированного управления (single system image), надежного функционирования (availability) и эффективного использования (performance). Кластеры могут быть образованы на базе уже существующих у потребителей отдельных компьютеров либо же сконструированы из типовых компьютерных элементов.

Применение кластеров может также в некоторой степени устранить проблемы, связанные с разработкой параллельных алгоритмов и программ, поскольку повышение вычислительной мощности отдельных процессоров позволяет строить кластеры из сравнительно небольшого количества (несколько десятков) отдельных компьютеров (lowly parallel processing). Тем самым, для параллельного выполнения в алгоритмах решения вычислительных задач достаточно выделять только крупные независимые части расчетов (coarse granularity), что, в свою очередь, снижает сложность построения параллельных методов вычислений и уменьшает потоки передаваемых данных между компьютерами кластера. Вместе с этим следует отметить, что организация взаимодействия вычислительных узлов кластера при помощи передачи сообщений обычно приводит к значительным временным задержкам, и это накладывает дополнительные ограничения на тип разрабатываемых параллельных алгоритмов и программ.

Моделирование и анализ параллельных вычислений

Для описания существующих информационных зависимостей в выбираемых алгоритмах решения задач может быть использована модель в виде графа "операции – операнды"

При построении модели будет предполагаться, что время выполнения любых вычислительных операций является одинаковым и равняется 1.

Передача данных между вычислительными устройствами выполняется мгновенно без каких-либо затрат времени .

Представим множество операций, выполняемых в исследуемом алгоритме решения вычислительной задачи, и существующие между операциями информационные зависимости в виде ациклического ориентированного графа

$G = (V, R)$,

где $V = \{1,\ldots,|V|\}$ есть множество вершин графа, представляющих выполняемые операции алгоритма, а R есть множество дуг графа (при этом дуга $r = (i, j)$ принадлежит графу только в том случае, если операция j использует результат выполнения операции i).

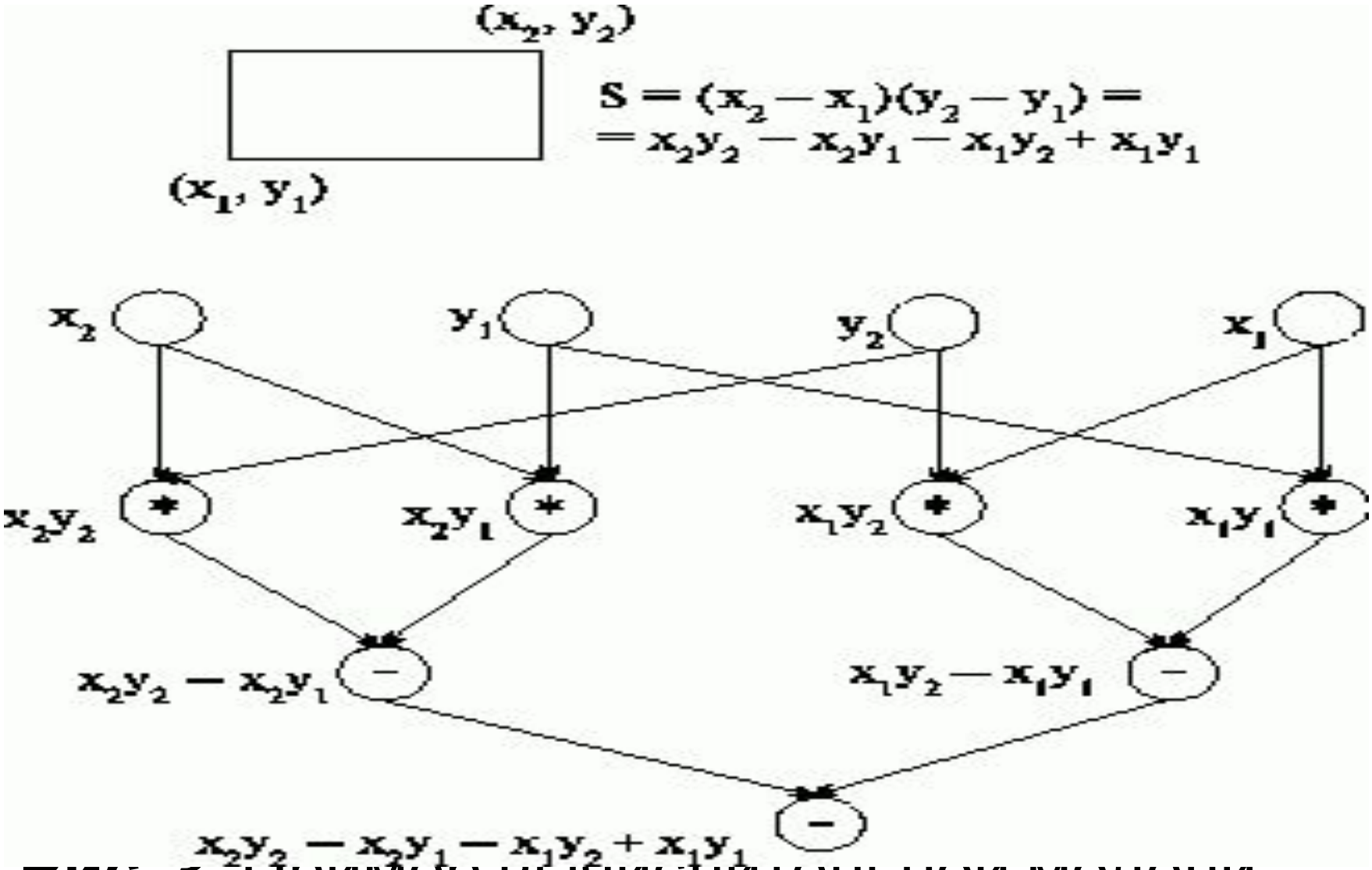


Рис. 3. Пример вычислительной модели алгоритма в виде графа "операции – операнды"

Описание схемы параллельного выполнения алгоритма

Операции алгоритма, между которыми нет пути в рамках выбранной схемы вычислений, могут быть выполнены параллельно (для вычислительной схемы на рис.3, например, параллельно могут быть реализованы сначала все операции умножения, а затем первые две операции вычитания). Возможный способ описания параллельного выполнения алгоритма может состоять в следующем

Пусть p есть количество процессоров, используемых для выполнения алгоритма. Тогда для параллельного выполнения вычислений необходимо задать множество (расписание)

$$H_p = \{(i, P_i, t_i) : i \in V\},$$

в котором для каждой операци$i \in V$ указывается номер используемого для выполнения операции процессора P_i и время начала выполнения операции t_i. Для того чтобы расписание было реализуемым, необходимо выполнение следующих требований при задании множества H :

$$\forall i, j \in V : t_i = t_j \Rightarrow P_i \neq P_j$$

, т.е. один и тот же процессор не должен назначаться разным операциям в один и тот же момент;

$$\forall (i, j) \in R t_j \geq t_i + 1$$

, т.е. к назначаемому моменту выполнения операции все необходимые данные уже должны быть вычислены.

Определение времени выполнения параллельного алгоритма

Вычислительная схема алгоритма G совместно с расписанием H_p может рассматриваться как модель параллельного алгоритма $A_p(G,H_p)$, исполняемого с использованием p процессоров. Время выполнения параллельного алгоритма определяется максимальным значением времени, применяемым в распи

$$T_p(G, H_p) = \max_{i \in V}(t_i + 1)$$

Для выбранной схемы вычислений желательно использование расписания, обеспечивающего минимальное время исполне

$$T_p(G) = \min_{H_p} T_p(G, H_p)$$

Уменьшение времени выполнения может быть обеспечено и путем подбора наилучшей вычислительной схемы

$$T_p = \min_G T_p(G)$$

Оценки $T_p(G,H_p)$, $T_p(G)$ и T_p могут быть применены в качестве показателей времени выполнения параллельного алгоритма. Кроме того, для анализа максимально возможного параллелизма можно определить оценку наиболее быстрого исполнения алгоритма

$$\hat{T}_\infty = \min_{p \geq 1} T_p$$

T_∞

Оценку можно рассматривать как минимально возможное время выполнения параллельного алгоритма при использовании неограниченного количества процессоров (концепция вычислительной системы с бесконечным количеством процессоров, обычно называемой **паракомпьютером**, широко применяется при теоретическом анализе

Оценка T_1 определяет время выполнения алгоритма при использовании одного процессора и представляет, тем самым, время выполнения последовательного варианта алгоритма решения задачи. Построение подобной оценки является важной задачей при анализе параллельных алгоритмов, поскольку она необходима для определения эффекта использования параллелизма (ускорения времени решения задачи). Очевидно, что

$$T_1(G) = |\overline{V}|$$

где $|\overline{V}|$, напомним, есть количество вершин вычислительной схемы без вершин ввода. Важно отметить, что если при определении оценки ограничиться рассмотрением только одного выбранного алгоритма решения задачи и использовать величину

$$T_1 = \min_G T_1(G)$$

то получаемые при такой оценке показатели ускорения будут характеризовать эффективность распараллеливания выбранного алгоритма. Для оценки эффективности параллельного решения исследуемой вычислительной задачи время последовательного решения следует определять с учетом различных последовательных алгоритмов, т.е. используя величину

$$T_1^* = \min T_1$$

где операция минимума берется по множеству всех возможных последовательных алгоритмов решения данной задачи

Теоретические положения, характеризующие свойства оценок времени выполнения параллельного алгоритма .

Теорема 1. Минимально возможное время выполнения параллельного алгоритма определяется длиной максимального пути вычислительной схемы алгоритма, т.е.

$$T_\infty(G) = d(G)$$

Теорема 2. Пусть для некоторой вершины вывода в вычислительной схеме алгоритма существует путь из каждой вершины ввода. Кроме того, пусть входная степень вершин схемы (количество входящих дуг) не превышает 2. Тогда минимально возможное время выполнения параллельного алгоритма ограничено снизу $T_\infty(G) = log_2 n$,

где n есть количество вершин ввода в схеме алгоритма.

Теорема 3. При уменьшении числа используемых процессоров время выполнения алгоритма увеличивается пропорционально величине уменьшения количества процессоров, т.е.

$$\forall p \Rightarrow T_p < T_\infty + T_1/p.$$

Теорема 4. Для любого количества используемых процессоров справедлива следующая верхняя оценка для времени выполнения параллельного алгоритма

$$\forall p \Rightarrow T_p < T_\infty + T_1/p$$

Теорема 5. Времени выполнения алгоритма, которое сопоставимо с минимально возможным временем T_∞, можно достичь при количестве процессоров $p \sim T_1/T_\infty$,

а именно, $p \geq T_1/T_\infty \Rightarrow T_p \leq 2T_\infty$

При меньшем количестве процессоров время выполнения алгоритма не может превышать более чем в 2 раза наилучшее время вычислений при имеющемся числе проц

$$p < T_1/T_\infty \Rightarrow \frac{T_1}{p} \leq T_p \leq 2\frac{T_1}{p}$$

Приведенные утверждения позволяют дать следующие рекомендации по правилам формирования параллельных алгоритмов:

- при выборе вычислительной схемы алгоритма должен использоваться граф с минимально возможным диаметром (см. теорему 1);
- для параллельного выполнения целесообразное количество процессоров определяется величиной $p \sim T_1/T_\infty$

(см. теорему 5);

- время выполнения параллельного алгоритма ограничивается сверху

Показатели эффективности параллельного алгоритма

Ускорение (**speedup**), получаемое при использовании параллельного алгоритма для p процессоров, по сравнению с последовательным вариантом выполнения вычислений определяется величиной

$$S_p(n)=T_1(n)/T_p(n),$$

т.е. как отношение времени решения задач на скалярной ЭВМ к времени выполнения параллельного алгоритма (величина n применяется для параметризации вычислительной сложности решаемой задачи и может пониматься, например, как количество входных данных задачи).

Эффективность (**efficiency**) использования параллельным алгоритмом процессоров при решении задачи определяется соотношением

$$E_p(n)=T_1(n)/(pT_p(n))=S_p(n)/p$$

(величина эффективности определяет среднюю долю времени выполнения алгоритма, в течение которой

Из приведенных соотношений можно показать, что в наилучшем случае $S_p(n)=p$ и $E_p(n)=1$. При практическом применении данных показателей для оценки эффективности параллельных вычислений следует учитывать два важных момента:

- При определенных обстоятельствах ускорение может оказаться больше числа используемых процессоров $S_p(n)>p$ - в этом случае говорят о существовании сверхлинейного (superlinear) ускорения. Несмотря на парадоксальность таких ситуаций (ускорение превышает число процессоров), на практике сверхлинейное ускорение может иметь место. Одной из причин такого явления может быть неодинаковость условий выполнения последовательной и параллельной программ

- Попытки повышения качества параллельных вычислений по одному из показателей (ускорению или эффективности) могут привести к ухудшению ситуации по другому показателю, ибо показатели качества параллельных вычислений являются часто противоречивыми. Например, повышение ускорения обычно может быть обеспечено за счет увеличения числа процессоров, что приводит к падению эффективности. И наоборот, повышение эффективности достигается во многих случаях при уменьшении числа процессоров (в предельном случае идеальная эффективность $e_p(n)=1$ легко обеспечивается при использовании одного процессора).

Как результат, разработка методов параллельных вычислений часто предполагает выбор некоторого компромиссного варианта с учетом желаемых показателей ускорения и эффективности

При выборе надлежащего параллельного способа решения задачи может оказаться полезной оценка **стоимости (cost)** вычислений, определяемой как произведение времени параллельного решения задачи и числа используемых процессоров

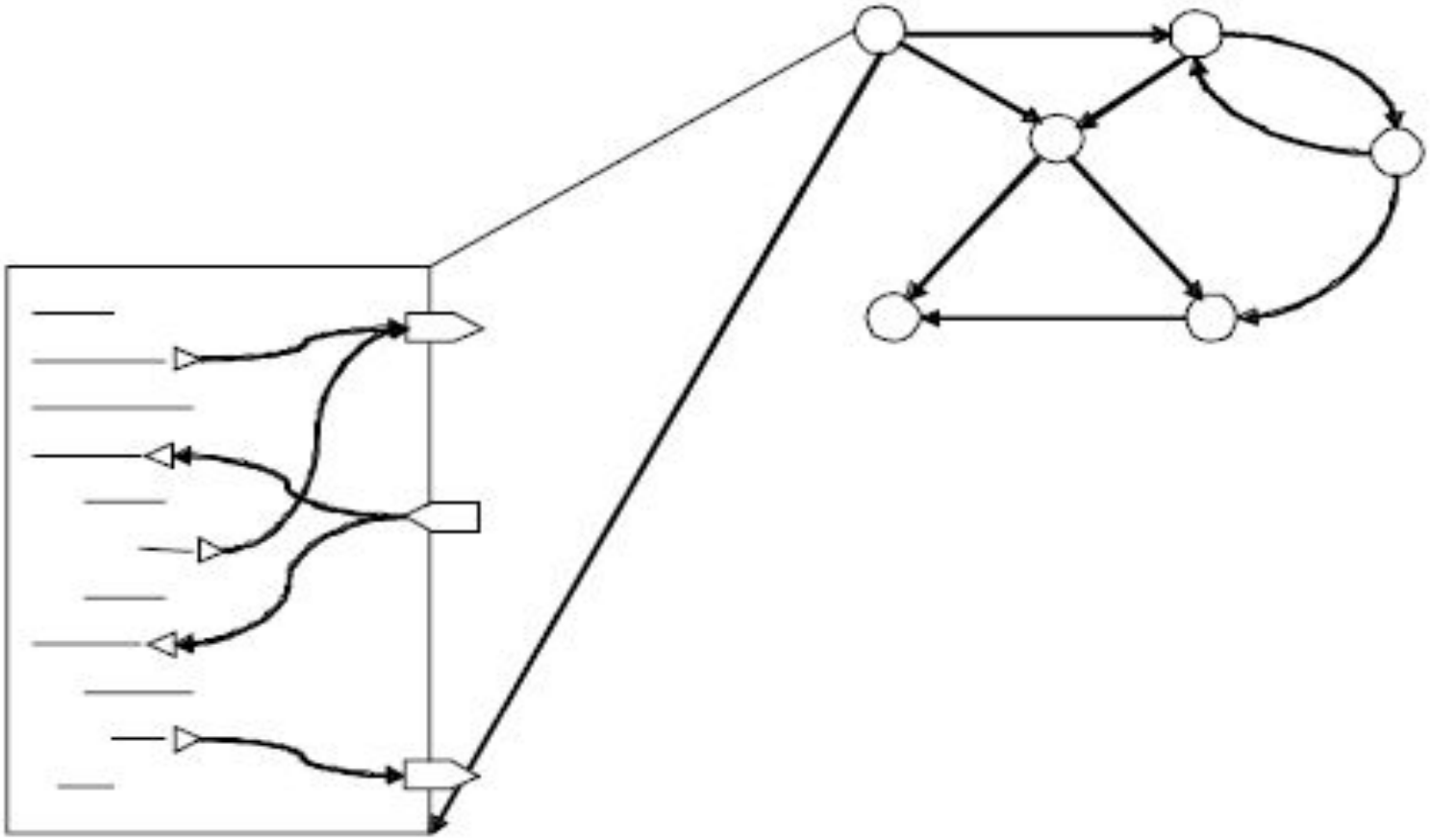
$C_p = pT_p$.

В связи с этим можно определить понятие **стоимостно-оптимального (cost-optimal)** параллельного алгоритма как метода, стоимость которого является пропорциональной времени выполнения наилучшего последовательного алгоритма.

Моделирование параллельных программ

На стадии проектирования параллельный метод может быть представлен в виде графа "подзадачи – сообщения" .

На стадии выполнения для описания параллельной программы может быть использована модель в виде графа "процессы – каналы", в которой вместо подзадач используется понятие процессов, а информационные зависимости заменяются каналами передачи сообщений. Дополнительно на этой модели может быть показано распределение процессов по процессорам вычислительной системы, если количество подзадач превышает число процессоров



◁ [▷] — операции приема (передачи)

⬅ [➡] — входные (выходные) каналы для взаимодействия процессов

Рис. 4. Модель параллельной программы в виде графа "процессы - каналы"

- под **процессом** будем понимать выполняемую на процессоре программу, которая использует для своей работы часть локальной памяти процессора и содержит ряд операций приема/передачи данных для организации информационного взаимодействия с другими выполняемыми процессами параллельной программы;
- **канал передачи данных** с логической точки зрения может рассматриваться как очередь сообщений, в которую один или несколько процессов могут отправлять пересылаемые данные и из которой процесс -адресат может извлекать сообщения, отправляемые другими процессами

Модели жизненного цикла для разработки программных систем За десятилетия опыта построения программных систем был наработан ряд типичных схем последовательности выполнения работ при проектировании и разработки ПС. Такие схемы получили название моделей ЖЦ.

Модель жизненного цикла - это схема выполнения работ и задач в рамках процессов, обеспечивающих разработку, эксплуатацию и сопровождение программного продукта, и отражающая эволюцию ПС, начиная от формулировки требований к ней до прекращения пользоваться ею

Исторически в эту схему работ включают:

- разработку требований или технического задания;
- разработку системы или технического проекта;
- программирование или рабочее проектирование;
- пробную эксплуатацию;
- сопровождение и улучшение;
- снятие с эксплуатации.

Основное назначение моделей ЖЦ состоит в следующем:

- планирование и распределение работ между разработчиками и ресурсов, а также управление программным проектом;
- обеспечение взаимодействия между разработчиками проекта и заказчиком;
- наблюдение и контроль работ, оценивание промежуточных продуктов ЖЦ на соблюдение спецификаций требований, правильное их выполнение, оценивание продукта и реальных затрат, в том числе и по применяемым программным средствам и инструментам;
- согласование промежуточных результатов с заказчиком;
- проверка правильности конечного продукта путем его тестирования на запланированных и согласованных с заказчиком наборах тестов;
- оценивание соответствия характеристик качества полученного продукта заданным требованиям;
- обсуждение используемых процессов ЖЦ в плане оценки их возможностей и недостатков, проявившихся при их применении, а также определение направлений усовершенствования или модернизации ЖЦ и др.

В связи с такими задачами, возлагаемыми на модель ЖЦ, необходимо сделать правильный выбор процессов, их задач и действий для построения модели ЖЦ ПС, которая удовлетворяет концептуальной идее проектируемой системы с учетом ее сложности и масштаба работ. В модель ЖЦ обязательно включаются процессы реализации работ и задач, обеспечивающие создание промежуточного продукта и переход к следующему процессу модели.

Процессы ЖЦ стандарта ISO/IEC 12207

При выборе схемы модели ЖЦ для конкретной предметной области, решаются вопросы включения важных для создаваемого продукта видов работ или не включения несущественных работ. На сегодня основой формирования новой модели ЖЦ для конкретной прикладной системы является стандарт ISO/IEC 12207, который задает полный набор процессов (более 40), охватывающий все возможные виды работ и задач, связанных с построением ПС, начиная с анализа предметной области и кончая изготовлением соответствующего продукта. Данный стандарт содержит основные и вспомогательные процессы (рис.1, рис.2).

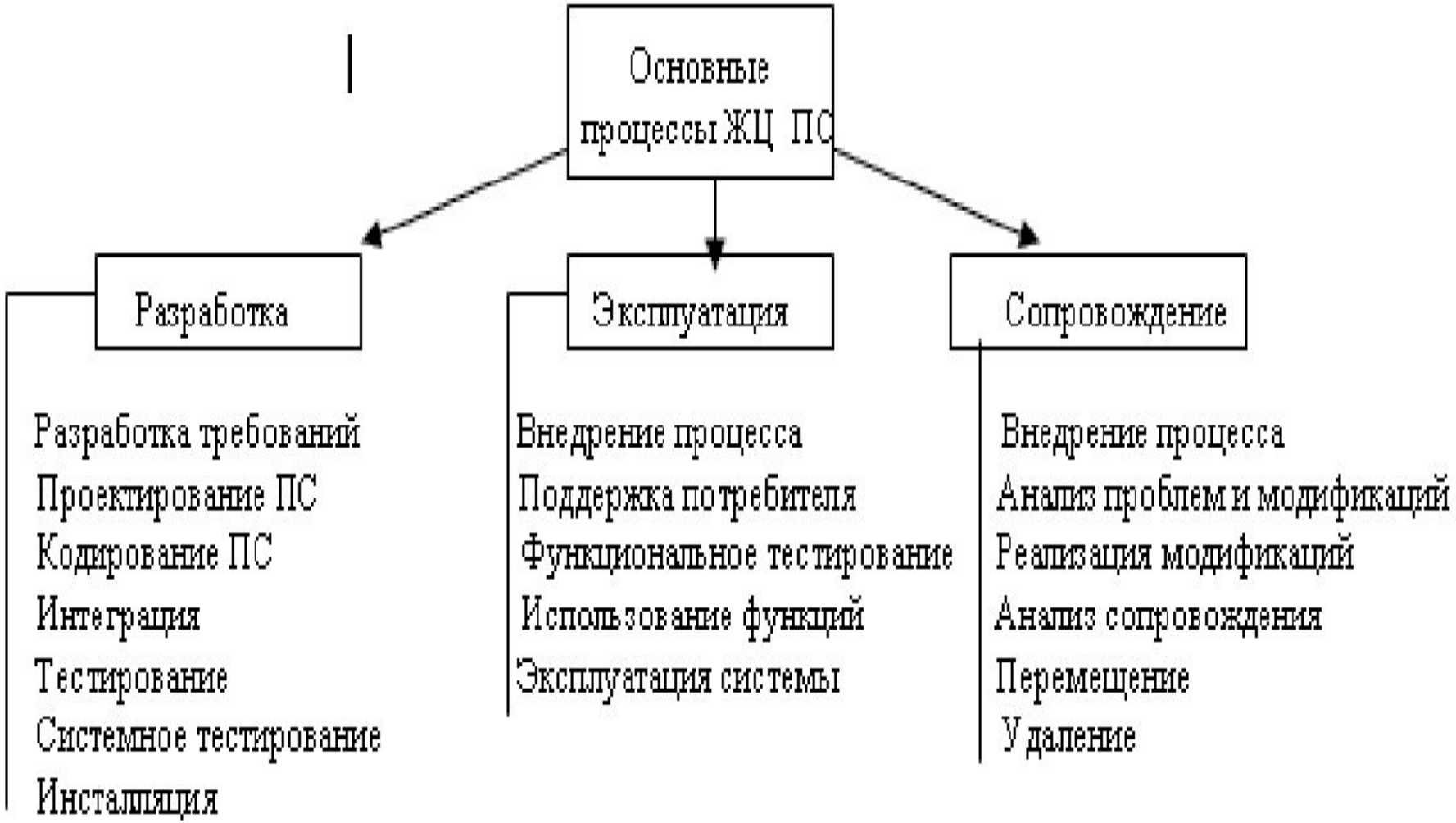


Рис. 1. Схема основных процессов ЖЦ ПС

Рис. 2. Схема вспомогательных процессов ЖЦ ПС

Являясь стандартом высокого уровня, он не задает детали того, как надо выполнять действия или задачи, составляющие процессы. Он также не задает требований к формату и содержанию документов, выпускаемых на разных процессах.

Процессы, действия и задачи приведены в стандарте в наиболее общей естественной последовательности. Это не значит, что в такой же последовательности они должны быть применены в конкретной модели ЖЦ ПС. В зависимости от проекта процессы, действия и задачи стандарта выбираются, упорядочиваются и включаются в модель ЖЦ. При применении они могут перекрывать, прерывать друг друга, выполняться итерационно или рекурсивно. Это определяет "динамический" характер стандарта и позволяет реализовать с его помощью произвольную модель ЖЦ ПС. Поэтому организации, которая намерена применить этот стандарт в своей работе, понадобятся дополнительные стандарты или процедуры, определяющие разные детали по применению выбранных элементов ЖЦ. Отметим, что комитет ISO выпускает руководства и процедуры, дополняющие стандарт 12207.

Процессы, включенные в модель ЖЦ, предназначены для реализации стандартных задач процессов ЖЦ и могут привлекать другие процессы для реализации специализированных задач системы (например, защиты данных). Интерфейсы (входы и выходы) любых двух процессов ЖЦ должны быть минимальными и каждый из них должен удовлетворять следующим правилам:

- если процесс А вызывается процессом В и только процессом В, то А принадлежит В ;
- если функция вызывается более чем одним процессом, то она становится отдельным процессом;
- проверка любой функции в ЖЦ является обязательной.

Характеристики качества программных средств

Международный стандарт ISO 9126:1-4 (1991г.), переиздан в 2002г. Метрики характеристик качества отражают:

- внешние качества – заданные требованиями заказчика в спецификациях и ...;
- внутренние качества – проявляются в процессе разработки и др. этапов жизненного цикла;
- качество при использовании вашего ПП в процессе эксплуатации и результативность

Модель характеристик качества состоит из 6 базовых показателей:

1. Функциональные возможности:
   - пригодность для применения по назначению;
   - корректность (правильность, точность) реализации требованиям;
   - способность к взаимодействию с компонентами и средой;
   - защищенность, безопасность функционирования.
2. Надежность ПП:
   - степень завершенности;
   - степень тестированности ;
   - устойчивость при наличии дефектов и ошибок;
   - восстанавливаемость после проявления дефектов (restart);
   - доступность, т.е. готовность реализации требуемых функций.
3. Эффективность ПП рекомендуется отражать:
   - субвременнóй эффективностью реализации программ;
   - используемостью вычислительных ресурсов.

4. Применимость (практичность) ПП:

- понятность функций и сопроводительной документации;
- простота использования комплекса программ;
- простота изучения процесса функционирования и освоения.

5. Сопровождаемость:

- анализируемость, т.е. удобство для анализа предложений модификации;
- изменяемость компонентов и всего комплекса программ;
- тестированность изменений при сопровождении.

6. Мобильность (переносимость):

- адаптируемость к изменениям среды;
- простота установки – инсталляций после переноса;
- замещаемость компонентов при корректировке программ.

Характеристики, субхарактеристики и атрибуты качества ПП имеют 3 уровня детализации:

- Категорийно-описательные – это описательные характеристики функций, категории ответственности защищенности и важности информационных потоков в вашей системе.
- Количественные показатели – измеряемые характеристики, измеряются часами, метрикой, номинальной шкалой.
- Качественные – несколько упорядоченных свойств, которые характеризуются некоторыми субъективными оценками: хорошо, плохо, да, нет.

Основные количественные характеристики программных средств и их атрибуты

Характеристики качества	**Мера**	**Шкала**
I. Надежность ПИ 1. Завершенность: - наработка на отказ при отсутствии рестарта - степень покрытия тестами функций и структуры программного изделия	Час % 	10-1000 50-100
2. Устойчивость - наработка на отказ при наличии автоматического рестарта - относительные ресурсы, отведенные на обеспечение надежности и рестарта	Час %	10-1000 10-90
3. Восстанавливаемость системы - длительность восстановления	Мин.	0.1 - 10
4. Доступность-готовность - относительное время работоспособности функционирования	вероятность	0.9-0.999
II. Эффективность 1. Временная эффективность - время отклика получения результата на типовое задание - пропускная способность - число типовых заданий в единицу времени	Сек. Число запросов в минуту	0.1-100 1-1000
2. Используемость ресурсов: - относительная величина используемости ресурсов ЭВМ, используемых программой	вероятность	0.7-0.95

Основные качественные характеристики программных средств и их аттрибутов

Характеристики	**Мера**	**Школа**	**Приоритет (1-10)**
Практичность 1. Понятность: - четкость концепции программных средств - демонстрационные возможности - наглядность и полнота документации	Порядковая	Отл. Хор. Удовл. Неуд.	1-10
2. Полнота использования - простота управления функциями программной системы - комфортность эксплуатации - среднее время ввода заданий - среднее время отлика на задание	Порядковая Сек. Сек	Отл. Хор. Удовл. Неуд. 1-1000 1-1000	1-10
3. Изучаемость - трудоемкость освоения ПС - продолжительность изучения - объем эксплуатационной документации - объем электронных учебников	Челокеко- часы Час Стр Кбайт	1-100 1-100 10-1000 100-10 00	

Характеристики	Мера	Школа	Приоритет (1-10)
4. Привлекательность			
- субъективные или экспертные оценки.			
Сопровождаемость 3. Анализируемость - стройность архитектуры программного средства - унифицированность интерфейса - полнота и корректность документации	Порядковая	Отл., хор., уд., неуд.	
4. Изменяемость - трудоемкость подготовки изменений - длительность подготовки изменений	Человеко- часы, Часы	1-1000 1-1000	
5. Стабильность - устойчивость при негативных проявлениях при изменениях	Порядковая	Отл,Хор,Уд, Неуд	
6. Тестируемость - трудоемкость тестирования изменений на этапе сопровождения - длительность тестирования изменения	Человеко- часы Часы	1-1000 1-100	

Характеристики	**Мера**	**Школа**	**Приоритет (1-10)**
Мобильность 1. Адаптируемость - трудоемкость адаптирования - длительность адаптации	 Чел-час Час	 1-100 1-100	
2. Простота установки: - трудоемкость установки - длительность установки	 Чел-час Час	 1-100 1-100	
3. Замещаемость - трудоемкость установки	Чел-час	1-100	

Метод установления правильности программ при помощи строгих средств известен как верификация программ.

В отличие от тестирования программ, где анализируются свойства отдельных процессов выполнения программы, верификация имеет дело со свойствами программ.

В основе метода верификации лежит предположение о том, что существует программная документация, соответствие которой требуется доказать. Документация должна содержать:

- спецификацию ввода-вывода (описание данных, не зависящих от процесса обработки);
- свойства отношений между элементами векторов состояний в выбранных точках программы;
- спецификации и свойства структурных подкомпонентов программы;
- спецификацию структур данных, зависящих от процесса обработки.

К методам проверки правильности программ относятся:

1) методы доказательство правильности программ;

2) верификация и аттестация программ.

Методы доказательства правильности программ, появились в 80–е годы, делятся на два класса:

1. Точные методы доказательства правильности программ.

2. Методы доказательства частичной правильности программ.

Наиболее известными точными методами доказательства программ являются метод рекурсивной индукции или индуктивных утверждений Флойда и Наура и метод структурной индукции Хоара и др. Эти методы основываются на утверждениях и пред и пост–

Метод Флойда и Наура основан на определении условий для входных и выходных данных и в выборе контрольных точек в доказываемой программе так, чтобы путь прохождения по программе пересекал хотя бы одну контрольную точку. Для этих точек формулируются утверждения о состоянии и значениях переменных в них (для циклов эти утверждения должны быть истинными при каждом прохождении цикла инварианта).

Каждая точка рассматривается для индуктивного утверждения того, что формула остается истинной при возвращении в эту точку программы и зависит не только от входных и выходных данных, но и от значений промежуточных переменных. На основе индуктивных утверждений и условий на аргументы создаются утверждения с условиями проверки правильности программы в отдельных ее точках. Для каждого пути программы между двумя точками устанавливается проверка на соответствие условий правильности и определяется истинность этих условий при успешном завершении программы на данных, удовлетворяющих входным условиям. Данный метод доказательства уменьшает число ошибок и время тестирования программы, обеспечивает отработку спецификаций программы на полноту, однозначность и

Метод Хоара - это усовершенствованный метод Флойда, основанный на аксиоматическом описании семантики языка программирования исходных программ. Каждая аксиома описывает изменение значений переменных с помощью операторов этого языка. Формализация операторов перехода и вызовов процедур обеспечивается с помощью правил вывода, содержащих индуктивные высказывания для каждой точки и функции исходной программы.

Система правил вывода дополняется механизмом переименования глобальных переменных, условиями на аргументы и результаты, а также на правильность задания данных программы. Оператор перехода трактуется как выход из циклов и аварийных ситуаций.

Описание с помощью системы правил утверждений - громоздкое и отличается неполнотой, поскольку все правила предусмотреть невозможно. Данный метод проверялся экспериментально на множестве

Таким образом, алгоритм доказательства правильности программы методом индуктивных высказываний представляется в следующем виде:

1) Построить структуру программы.

2) Выписать входное и выходное высказывания.

3) Сформулировать для всех циклов индуктивные высказывания.

4) Составить список выделенных путей.

5) Построить условия верификации.

6) Доказать условие верификации.

7) Доказать, что выполнение программы закончится.

Этот метод сравним с обычным процессом чтения текста программы (метод сквозного контроля). Различие заключается в степени формализации.

Метод Маккарти состоит в структурной проверке функций, работающих над структурными типами данных, изменяет структуры данных и диаграммы перехода во время символьного выполнения программ. Эта техника включает в себя моделирование выполнения кода с использованием символов для изменяемых данных. Тестовая программа имеет входное состояние, данные и условия ее выполнения.

Выполняемая программа рассматривается как серия изменений состояний. Самое последнее состояние программы считается выходным состоянием и если оно получено, то программа считается правильной. Данный метод обеспечивает высокое качество исходного кода.

Метод Дейкстры предлагает два подхода к доказательству правильности программ. Первый подход основан на модели вычислений, оперирующей с историями результатов вычислений программы, анализом путей прохождения и правил обработки большого объема информации. Второй подход базируется на формальном исследовании текста программы с помощью предикатов первого порядка. В процессе выполнения программа получает некоторое состояние, которое запоминается для дальнейших сравнений.

Основу метода составляет математическая индукция, абстрактное описание программы и ее вычисление. Математическая индукция применяется при прохождении циклов и рекурсивных процедур, а также необходимых и достаточных условий утверждений. Абстракция позволяет сформулировать некоторые количественные ограничения. При вычислении на основе инвариантных отношений проверяются на