Оглавление

[1. Понятие вычислительной системы 3](#_Toc535440680)

[2. Функции вычислительной системы 4](#_Toc535440681)

[3. Сфера применения многопроцессорных вычислительных систем 5](#_Toc535440682)

[4. Классификация вычислительных систем 12](#_Toc535440683)

[5. Характеристики многопроцессорных вычислительных систем (повышенная производительность, масштабируемость, минимально допустимое время простоя) 16](#_Toc535440684)

[6. Требования к компонентам МВС (отношение «стоимость/производительность», надежность и отказоустойчивость системы; масштабируемость системы; совместимость программного обеспечения) 22](#_Toc535440685)

[7. <>Операционные системы МВС. Коммуникации процессов. 28](#_Toc535440686)

[8. Функции ОС. Процесс. Поток. Синхронизация параллельных процессов. 29](#_Toc535440687)

[9. Классификация архитектур по параллельной обработке данных (SISD, MISD, SIMD, MIMD). 32](#_Toc535440688)

[10. Распределённые операционные системы. Классификация мультипроцессорных операционных систем. 36](#_Toc535440689)

[11. Идея распараллеливания вычислений. Типы параллелизма. 38](#_Toc535440690)

[12. SMP-архитектура. 41](#_Toc535440691)

[13. MPP-архитектура. 44](#_Toc535440692)

[14. PVP-архитектура. 46](#_Toc535440693)

[15. Составные коммутаторы. Коммутатор Клоза. Баньян-сети. Распределённые составные коммутаторы. 48](#_Toc535440694)

[16. Простые коммутаторы. Шинная архитектура. Разрешение конфликтов. Алгоритмы арбитража. 51](#_Toc535440695)

[17. Ассоциативные процессоры. 53](#_Toc535440696)

[18. <>Непредсказуемость производительности. Аспекты масштабирования. Совместимость и мобильность программного обеспечения. Недолговечность архитектур. 55](#_Toc535440697)

[19. <>Кластерные технологии. Архитектуры кластеров. 56](#_Toc535440698)

[20. <>ГРИД. Типы ГРИД-систем. 57](#_Toc535440699)

# 1. Понятие вычислительной системы

Это было самое объёмное объяснение, если кому понадобится добавить воды при сдаче. На всякий случай вот пару источников (но там говорится то же самое):

<https://studopedia.org/2-35770.html>

<https://sites.google.com/site/informatika1011kl/lekcia-vycislitelnye-sistemy>

В связи с кризисом классической структуры ЭВМ дальнейшее поступательное развитие вычислительной техники напрямую связано с переходом к параллельным вычислениям, с идеями построения многопроцессорных систем и сетей, объединяющих большое количество отдельных процессоров и (или) ЭВМ. Здесь появляются огромные возможности совершенствования средств вычислительной техники. Но следует отметить, что при несомненных практических достижениях в области параллельных вычислений, до настоящего времени отсутствует их единая теоретическая база.

Термин вычислительная система появился в начале - середине 60-х гг. при появлении ЭВМ III поколения. Это время знаменовалось переходом на новую элементную базу - интегральные схемы. Следствием этого явилось появление новых технических решений: разделение процессов обработки информации и ее ввода-вывода, множественный доступ и коллективное использование вычислительных ресурсов в пространстве и во времени. Появились сложные режимы работы ЭВМ - многопользовательская и многопрограммная обработка.

Под вычислительной системой (ВС) понимают совокупность взаимосвязанных и взаимодействующих процессоров или ЭВМ, периферийного оборудования и программного обеспечения, предназначенную для сбора, хранения, обработки и распределения информации.

# 2. Функции вычислительной системы

Тут я не нашёл адекватных материалов, воображение вам в руки

# 3. Сфера применения многопроцессорных вычислительных систем

В настоящее время сфера применения многопроцессорных вычислительных систем (МВС) непрерывно расширяется, охватывая все новые области в различных отраслях науки, бизнеса и производства. Стремительное развитие кластерных систем создает условия для использования многопроцессорной вычислительной техники в реальном секторе экономики.

Если традиционно МВС применялись в основном в научной сфере для решения вычислительных задач, требующих мощных вычислительных ресурсов, то сейчас из-за бурного развития бизнеса резко возросло количество компаний, отводящих использованию компьютерных технологий и электронного документооборота главную роль. В связи с этим непрерывно растет потребность в построении централизованных вычислительных систем для критически важных приложений, связанных с обработкой транзакций, управлением базами данных и обслуживанием телекоммуникаций. Можно выделить две основные сферы применения описываемых систем: обработка транзакций в режиме реального времени (OLTP, on-line transaction processing) и создание хранилищ данных для организации систем поддержки принятия решений (Data Mining, Data Warehousing, Decision Support System). Система для глобальных корпоративных вычислений — это, прежде всего, централизованная система, с которой работают практически все пользователи в корпорации, и, соответственно, она должна все время находиться в рабочем состоянии. Как правило, решения подобного уровня устанавливают в компаниях и корпорациях, где даже кратковременные простои сети могут привести к громадным убыткам. Поэтому для организации такой системы не подойдет обыкновенный сервер со стандартной архитектурой, вполне пригодный там, где нет жестких требований к производительности и времени простоя. Высокопроизводительные системы для глобальных корпоративных вычислений должны отличаться такими характеристиками как повышенная производительность, масштабируемость, минимально допустимое время простоя.

Наряду с расширением области применения по мере совершенствования МВС происходит усложнение и увеличение количества задач в областях, традиционно использующих высокопроизводительную вычислительную технику. В настоящее время выделен круг фундаментальных и прикладных проблем, эффективное решение которых возможно только с использованием сверхмощных вычислительных ресурсов. Этот круг, обозначаемый понятием "Grand challenges", включает следующие задачи:

* предсказания погоды, климата и глобальных изменений в атмосфере;
* науки о материалах;
* построение полупроводниковых приборов;
* сверхпроводимость;
* структурная биология;
* разработка фармацевтических препаратов;
* генетика;
* квантовая хромодинамика;
* астрономия;
* транспортные задачи;
* гидро- и газодинамика;
* управляемый термоядерный синтез;
* эффективность систем сгорания топлива;
* геоинформационные системы;
* разведка недр;
* наука о мировом океане;
* распознавание и синтез речи;
* распознавание изображений.

Многопроцессорные вычислительные системы могут существовать в различных конфигурациях. Наиболее распространенными типами МВС являются:

* системы высокой надежности;
* системы для высокопроизводительных вычислений;
* многопоточные системы.

Отметим, что границы между этими типами МВС до некоторой степени размыты, и часто система может иметь такие свойства или функции, которые выходят за рамки перечисленных типов. Более того, при конфигурировании большой системы, используемой как система общего назначения, приходится выделять блоки, выполняющие все перечисленные функции.

МВС являются идеальной схемой для повышения надежности информационно-вычислительной системы. Благодаря единому представлению, отдельные узлы или компоненты МВС могут незаметно для пользователя заменять неисправные элементы, обеспечивая непрерывность и безотказную работу даже таких сложных приложений как базы данных.

Катастрофоустойчивые решения создаются на основе разнесения узлов многопроцессорной системы на сотни километров и обеспечения механизмов глобальной синхронизации данных между такими узлами.

МВС для высокопроизводительных вычислений предназначены для параллельных расчетов. Имеется много примеров научных расчетов, выполненных на основе параллельной работы нескольких недорогих процессоров, обеспечивающих одновременное проведение большого числа операций.

МВС для высокопроизводительных вычислений обычно собраны из многих компьютеров. Разработка таких систем – процесс сложный, требующий постоянного согласования таких вопросов как инсталляция, эксплуатация и одновременное управление большим числом компьютеров, технических требований параллельного и высокопроизводительного доступа к одному и тому же системному файлу (или файлам), межпроцессорной связи между узлами и координации работы в параллельном режиме. Эти проблемы проще всего решаются при обеспечении единого образа операционной системы для всего кластера. Однако реализовать подобную схему удается далеко не всегда, и обычно она применяется лишь для небольших систем.

Многопоточные системы используются для обеспечения единого интерфейса к ряду ресурсов, которые могут со временем произвольно наращиваться (или сокращаться). Типичным примером может служить группа web-серверов.

Главной отличительной особенностью многопроцессорной вычислительной системы является ее производительность, т.е. количество операций, производимых системой за единицу времени. Различают пиковую и реальную производительность. Под пиковой понимают величину, равную произведению пиковой производительности одного процессора на число таких процессоров в данной машине. При этом предполагается, что все устройства компьютера работают в максимально производительном режиме. Пиковая производительность компьютера вычисляется однозначно, и эта характеристика является базовой, по которой производят сравнение высокопроизводительных вычислительных систем . Чем больше пиковая производительность, тем (теоретически) быстрее пользователь сможет решить свою задачу. Пиковая производительность есть величина теоретическая и, вообще говоря, недостижимая при запуске конкретного приложения. Реальная же производительность, достигаемая на данном приложении, зависит от взаимодействия программной модели, в которой реализовано приложение, с архитектурными особенностями машины, на которой приложение запускается.

Существует два способа оценки пиковой производительности компьютера. Один из них опирается на число команд, выполняемых компьютером за единицу времени. Единицей измерения, как правило, является MIPS (Million Instructions Per Second). Производительность, выраженная в MIPS, говорит о скорости выполнения компьютером своих же инструкций. Но, во-первых, заранее не ясно, в какое количество инструкций отобразится конкретная программа, а во-вторых, каждая программа обладает своей спецификой, и число команд от программы к программе может меняться очень сильно. В связи с этим данная характеристика дает лишь самое общее представление о производительности компьютера.

Другой способ измерения производительности заключается в определении числа вещественных операций, выполняемых компьютером за единицу времени. Единицей измерения является Flops (Floating point operations per second) – число операций с плавающей точкой, производимых компьютером за одну секунду. Такой способ является более приемлемым для пользователя, поскольку ему известна вычислительная сложность программы, и, пользуясь этой характеристикой, пользователь может получить нижнюю оценку времени ее выполнения.

Однако пиковая производительность получается только в идеальных условиях, т.е. при отсутствии конфликтов при обращении к памяти при равномерной загрузке всех устройств. В реальных условиях на выполнение конкретной программы влияют такие аппаратно-программные особенности данного компьютера как: особенности структуры процессора, системы команд, состав функциональных устройств, реализация ввода/вывода, эффективность работы компиляторов.

Одним из определяющих факторов является время взаимодействия с памятью, которое определяется ее строением, объемом и архитектурой подсистем доступа в память. В большинстве современных компьютеров в качестве организации наиболее эффективного доступа к памяти используется так называемая многоуровневая иерархическая память. В качестве уровней используются регистры и регистровая память, основная оперативная память, кэш-память, виртуальные и жесткие диски, ленточные роботы. При этом выдерживается следующий принцип формирования иерархии: при повышении уровня памяти скорость обработки данных должна увеличиваться, а объем уровня памяти – уменьшаться. Эффективность использования такого рода иерархии достигается за счет хранения часто используемых данных в памяти верхнего уровня, время доступа к которой минимально. А поскольку такая память обходится достаточно дорого, ее объем не может быть большим. Иерархия памяти относится к тем особенностям архитектуры компьютеров, которые имеют огромное значение для повышения их производительности.

Для того чтобы оценить эффективность работы вычислительной системы на реальных задачах, был разработан фиксированный набор тестов. Наиболее известным из них является LINPACK – программа, предназначенная для решения системы линейных алгебраических уравнений с плотной матрицей с выбором главного элемента по строке. LINPACK используется для формирования списка Top500 – пятисот самых мощных компьютеров мира. Однако LINPACK имеет существенный недостаток: программа распараллеливается, поэтому невозможно оценить эффективность работы коммуникационного компонента суперкомпьютера.

В настоящее время большое распространение получили тестовые программы, взятые из разных предметных областей и представляющие собой либо модельные, либо реальные промышленные приложения. Такие тесты позволяют оценить производительность компьютера действительно на реальных задачах и получить наиболее полное представление об эффективности работы компьютера с конкретным приложением.

Наиболее распространенными тестами, построенными по этому принципу, являются: набор из 24 Ливерморских циклов (The Livermore Fortran Kernels, LFK) и пакет NAS Parallel Benchmarks (NPB), в состав которого входят две группы тестов, отражающих различные стороны реальных программ вычислительной гидродинамики. NAS тесты являются альтернативой LINPACK, поскольку они относительно просты и в то же время содержат значительно больше вычислений, чем, например, LINPACK или LFK.

Однако при всем разнообразии тестовые программы не могут дать полного представления о работе компьютера в различных режимах. Поэтому задача определения реальной производительности многопроцессорных вычислительных систем остается пока нерешенной.

# 4. Классификация вычислительных систем

Большое разнообразие вычислительных систем породило естественное желание ввести для них какую-то классификацию. Эта классификация должна однозначно относить ту или иную вычислительную систему к некоторому классу, который, в свою очередь, должен достаточно полно ее характеризовать. Таких попыток предпринималось множество. Одна из первых классификаций, ссылки на которую наиболее часто встречаются в литературе, была предложена М. Флинном в конце 60-х годов прошлого века. Она базируется на понятиях двух потоков: команд и данных. На основе числа этих потоков выделяется четыре класса архитектур:

* **SISD** (Single Instruction Single Data) - единственный поток команд и единственный поток данных. По сути дела это классическая машина фон Неймана. К этому классу относятся все однопроцессорные системы.
* **SIMD** (Single Instruction Multiple Data) - единственный поток команд и множественный поток данных. Типичными представителями являются матричные компьютеры, в которых все процессорные элементы выполняют одну и ту же программу, применяемую к своим (различным для каждого ПЭ) локальным данным. Некоторые авторы к этому классу относят и векторно-конвейерные компьютеры, если каждый элемент вектора рассматривать как отдельный элемент потока данных.
* **MISD** (Multiple Instruction Single Date) - множественный поток команд и единственный поток данных. М. Флинн не смог привести ни одного примера реально существующей системы, работающей на этом принципе. Некоторые авторы в качестве представителей такой архитектуры называют векторно-конвейерные компьютеры, однако такая точка зрения не получила широкой поддержки.
* **MIMD** (Multiple Instruction Multiple Date) - множественный поток команд и множественный поток данных. К этому классу относятся практически все современные многопроцессорные системы.

Поскольку в этой классификации все современные многопроцессорные системы принадлежат одному классу, то вряд ли эта классификация представляет сегодня какую-либо практическую ценность.

А теперь понеслись дополнительные классификации

По назначению вычислительные системы делят на **универсальные и специализированные.** Универсальные ВС предназначаются для ре­шения самых различных задач. Специализированные системы ориен­тированы на решение узкого класса задач.

По типу вычислительные системы можно разделить на **многомашинные и многопроцессорные ВС**. Исторически многомашинные вычислительные системы (ММС) появились первыми. Уже при использо­вании ЭВМ первых поколений возникали задачи повышения производительности, надежности и достоверности вычислений. Здесь возможны две ситуации:

* обе машины решают одну и ту же задачу и периодически сверяют результаты решения. Тем самым обеспечивался режим повышенной достоверности, уменьшалась вероятность появления ошибок в результатах вычислений.
* обе машины работают параллельно, но обрабатывают собственные потоки заданий. Возможность обмена информацией между машинами сохраняется. Этот вид работы относится к режиму повышенной производительности.

Многопроцессорные системы (МПС) строятся при комплексировании нескольких процессоров. В качестве общего ресурса они имеют общую оперативную память (ООП). Параллельная работа процессоров и использование ООП обеспечивается под управлением единой операционной системы.

По типу ЭВМ или процессоров, используемых для построения ВС, различают **однородные и неоднородные системы**. Однородные системы предполагают комплексирование однотипных ЭВМ (процессоров), неоднородные — разнотипных. В однородных системах значительно упрощается разработка и обслуживание технических и программных (в основном ОС) средств. В них обеспечивается возможность стандартизации и унификации соединений и процедур взаимодействия элементов системы. Упрощается обслуживание систем, облегчается модернизация и их развитие. Вместе с тем существуют и неоднородные ВС, в которых комплексируемые элементы очень сильно отличаются по своим техническим и функциональным характеристикам.

По степени территориальной разобщенности вычислительных модулей ВС делятся на **системы совмещенного (сосредоточенного) и распределенного (разобщенного) типов**. Обычно такое деление касается только ММС. Многопроцессорные системы относятся к системам совмещенного типа.

По методам управления элементами ВС различают **централизованные, децентрализованные и со смешанным управлением**. Помимо параллельных вычислений, производимых элементами системы, необходимо выделять ресурсы на обеспечение управления этими вычислениями. В централизованных ВС за это отвечает главная, или диспетчерская, ЭВМ (процессор). Ее задачей является распределение нагрузки между элементами, выделение ресурсов, контроль состояния ресурсов, координация взаимодействия. В децентрализованных системах функции уп­равления распределены между ее элементами. Каждая ЭВМ (процессор) системы сохраняет известную автономию. В системах со смешанным управлением совмещаются процедуры централизованного и децентрализованного управления. Перераспределение функций осуществляется в ходе вычислительного процесса, исходя из сложившейся ситуации.

По принципу закрепления вычислительных функций за отдельными ЭВМ (процессорами) различают **системы с жестким и плавающим закреплением функций**. В зависимости от типа ВС следует решать задачи статического или динамического размещения программных модулей и массивов данных, обеспечивая необходимую гибкость системы и надежность ее функционирования.

По режиму работы ВС различают **системы, работающие в оперативном и неоперативном временных режимах**. Первые, как правило, используют режим реального масштаба времени. Этот режим характеризуется жесткими ограничениями на время решения задач в системе и предполагает высокую степень автоматизации процедур ввода-вывода и обработки данных.

# 5. Характеристики многопроцессорных вычислительных систем (повышенная производительность, масштабируемость, минимально допустимое время простоя)

**Производительность**

Быстродействие вычислительного устройства определяется количеством выполняемых в секунду операций. Эта характеристика в большей степени относится к оценке скорости выполнения операций процессором. Список выполняемых операций разнообразен, существуют сложные операции, выполняемые за более длительное время, чем элементарные. Среднее быстродействие вычислительного устройства, с учетом операций различной сложности, характеризуется значением:

Vср= 100% / (Pi \* ti),

где Pi -процент операций i-го типа, определяемый классом выполняемых алгоритмов, а ti -время выполнения операции i-го типа, определяемое способом реализации операций в вычислительном устройстве. То есть производительность определяется операционными ресурсами устройства. Повышение производительности связывают прежде всего с быстродействием процессора.

Для оценки быстродействия процессоров применяют обычно тестовые наборы, оценивающие производительность относительно производительности процессора, принятого за талон. Для сравнения быстродействия процессоров применяются различные способы измерения.

Для сравнения процессоров с близкой архитектурой применяется показатель, равный усредненному числу операций, выполняемых за единицу времени.

Чтобы по возможности не привязываться к быстродействию памяти, выбираются обычно операции с внутренними регистрами процессоров.

На таком принципе оценки построена единица измерения MIPS (Mega Instructions Per Second), определяющая число миллионов выполненных инструкций в секунду. Во времена процессоров 8086/88 и 80286 их производительность была пропорциональна тактовой частоте. В последующих моделях процессоров стали появляться довольно значительные изменения архитектуры, в результате которых тактовая частота уже перестала быть единственным фактором, достоверно определяющим производительность.

Для измерения быстродействия при выполнении вычислений с плавающей точкой вместо единицы MIPS применяется единица FLOPS (FLoating point Operations Per Second) со всеми возможными десятичными приставками - MFLOPS, GFLOPS, TFLOPS.

Для сравнения быстродействия 32-битных процессоров с архитектурой 80x86 фирма Intel в 1992 году предложила свою единицу измерения - iCOMP (Intel COmparative Microprocessor Performance). Эти измерения позволили дать конечному пользователю упрощенный способ определения относительной мощности процессора. Для вычисления быстродействия определялась скорость выполнения некоторой смеси 16- и 32-битных операций четырех категорий: целочисленных, с плавающей точкой, скорости обработки графики и видео. Полученные относительные скорости вошли в общей показатель с определенными весовыми коэффициентами. В качестве базового процессора, по отношению к которому определяются относительные скорости принимают процессор с самой высокой производительностью на данный момент, его индекс равен 100. В 1996 году была введена новая единица - iCOMP Index 2.0, отличающаяся набором показателей и весовыми коэффициентами, а также выбором базового процессора. Из тестовой смеси исключили 16-битные инструкции (их вес в прежнем индексе составлял 70%). Кроме того, в новый показатель стал входить мультимедийный тест.

Еще один способ сравнения быстродействия процессоров -введение P-рейтинга.

P-рейтинг — это система оценки быстродействия процессора. Фирма Intel маркировала свои одноядерные (с одним набором исполнительных устройств) процессоры значением тактовой частоты. Для определения быстродействия своих процессоров фирмы AMD, Cyrics, IBM и SGS-Thompson ввели понятие P-Rating. Оно основано на сравнении быстродействия данного процессора с быстродействием процессора Pentium. Формально рейтинг процессоров других фирм соответствовал тактовой частоте, на которой должен был работать процессор для достижения аналогичной производительности процессора Pentium при выполнении приложений.

Маркировка AMD 5x86-P75 примечательна тем, что в ней впервые был использован P-рейтинг, который должен был подчеркнуть, что производительность процессора эквивалентна производительности процессора Pentium с тактовой частотой 75 МГц.

Процессор AMD Athlon 64 3800+ (имеет P- рейтинг = 3800) реально работает на частоте 2.6 ГГц, однако имеет производительность в приложениях сравнимую с процессором фирмы Intel с частотой 3.8 ГГц. Поэтому при выборе процессора важно просмотреть результаты сравнительного тестирования работы процессоров при выполнении различных приложений.

Как при этом исключить влияние архитектуры вычислительной системы?

Одним из основных вопросов при проведении сравнительного тестирования различных по архитектуре процессоров является выбор компонентов, способных обеспечить объективность результатов тестирования. Результаты тестов чистого процессорного быстродействия не должны зависеть от аппаратного окружения и применяемой операционной системы, поэтому тестирование производится с использованием эталонной системы.

Основой эталонной системы является системная плата, позволяющая устанавливать как процессоры Pentium, так и процессоры испытуемого класса. Эталонным тестом является пакет WinStone фирмы Ziff-Davis, тестирующий производительность работы текстовых процессоров, электронных таблиц, баз данных и деловой графики в среде Windows. Но тесты системного уровня, к числу которых относятся пакеты WinStone, предназначены для сравнения производительности готовых компьютерных систем и могут применяться для сравнительного тестирования процессоров, лишь при соблюдении идентичности всех параметров системы.

**Масштабируемость**

стырено с вики, более общих текстов без лишних понтов не найти

Масштаби́руемость (англ. scalability) — в электронике и информатике означает способность системы, сети или процесса справляться с увеличением рабочей нагрузки (увеличивать свою производительность) при добавлении ресурсов (обычно аппаратных).

Масштабируемость — важный аспект электронных систем, программных комплексов, систем баз данных, маршрутизаторов, сетей и т. п., если для них требуется возможность работать под большой нагрузкой. Система называется масштабируемой, если она способна увеличивать производительность пропорционально дополнительным ресурсам. Масштабируемость можно оценить через отношение прироста производительности системы к приросту используемых ресурсов. Чем ближе это отношение к единице, тем лучше. Также под масштабируемостью понимается возможность наращивания дополнительных ресурсов без структурных изменений центрального узла системы.

В системе с плохой масштабируемостью добавление ресурсов приводит лишь к незначительному повышению производительности, а с некоторого «порогового» момента добавление ресурсов не даёт никакого полезного эффекта.

**Время простоя**

В последние годы в литературе по вычислительной технике все чаще употребляется термин “системы высокой готовности” (High Availability Systems). Все типы систем высокой готовности имеют общую цель — минимизацию времени простоя. Имеется два типа времени простоя компьютера: плановое и неплановое. Минимизация каждого из них требует различной стратегии и технологии. Плановое время простоя обычно включает время, принятое руководством, для проведения работ по модернизации системы и для ее обслуживания. Неплановое время простоя является результатом отказа системы или компонента. Хотя системы высокой готовности, возможно, больше ассоциируются с минимизацией неплановых простоев, они оказываются также полезными для уменьшения планового времени простоя. Следует отметить, что высокая готовность не дается бесплатно. Стоимость систем высокой готовности намного превышает стоимость обычных систем.

Главными характеристиками систем высокой готовности по сравнению со стандартными системами являются пониженная частота отказов и более быстрый переход к нормальному режиму функционирования после возникновения неисправности посредством быстрого восстановления приложений и сетевых сессий до того состояния, в котором они находились в момент отказа системы. Следует отметить, что во многих случаях пользователей вполне может устроить даже небольшое время простоя в обмен на меньшую стоимость системы высокой готовности по сравнению со значительно более высокой стоимостью обеспечения режима непрерывной готовности.

# 6. Требования к компонентам МВС (отношение «стоимость/производительность», надежность и отказоустойчивость системы; масштабируемость системы; совместимость программного обеспечения)

**Отношение стоимость/производительность**

Добиться дополнительного повышения производительности в МВС тяжелее, чем произвести масштабирование внутри узла. Основным барьером является трудность организации эффективных межузловых связей. Коммуникации, которые происходят между узлами, должны быть устойчивы к большим задержкам программно поддерживаемой когерентности. Приложения с большим количеством взаимодействующих процессов работают лучше на основе SMP-узлов, в которых коммуникационные связи более быстрые. В кластерах, как и в МРР системах, масштабирование приложений более эффективно при уменьшении объема коммуникаций между процессами, работающими в разных узлах. Это обычно достигается путем разбиения данных. Именно такой подход используется в наиболее известном приложении на основе кластеров OPS (Oracle Parallel Server). Появление любого нового направления в вычислительной технике определяется требованиями компьютерного рынка. Поэтому у разработчиков компьютеров нет одной единственной цели. Большая универсальная вычислительная машина (мейнфрейм) или суперкомпьютер стоят дорого. Для достижения поставленных целей при проектировании высокопроизводительных конструкций приходится игнорировать стоимостные характеристики.

Суперкомпьютеры фирмы Cray Research и высокопроизводительные мейнфреймы компании IBM относятся именно к этой категории компьютеров. Другим крайним примером может служить низкостоимостная конструкция, где производительность принесена в жертву для достижения низкой стоимости. К этому направлению относятся персональные компьютеры различных клонов IBM PC. Между этими двумя крайними направлениями находятся конструкции, основанные на отношении стоимость/производительность, в которых разработчики находят баланс между стоимостными параметрами и производительностью. Типичными примерами такого рода компьютеров являются миникомпьютеры и рабочие станции. Для сравнения различных компьютеров между собой обычно используются стандартные методики измерения производительности. Эти методики позволяют разработчикам и пользователям использовать полученные в результате испытаний количественные показатели для оценки тех или иных технических решений, и, в конце концов, именно производительность и стоимость дают пользователю рациональную основу для решения вопроса, какой компьютер выбрать.

**Масштабируемость**

Масштабируемость представляет собой возможность наращивания числа и мощности процессоров, объемов оперативной и внешней памяти и других ресурсов вычислительной системы. Масштабируемость должна обеспечиваться архитектурой и конструкцией компьютера, а также соответствующими средствами программного обеспечения.

Так, например, возможность масштабирования кластера ограничена значением отношения скорости процессора к скорости связи, которое не должно быть слишком большим (реально это отношение для больших систем не может быть более 3-4, в противном случае не удается даже реализовать режим единого образа операционной системы). С другой стороны, последние 10 лет истории развития процессоров и коммуникаторов показывают, что разрыв по скорости между ними все увеличивается. Добавление каждого нового процессора в действительно масштабируемой системе должно давать прогнозируемое увеличение производительности и пропускной способности при приемлемых затратах. Одной из основных задач при построении масштабируемых систем является минимизация стоимости расширения компьютера и упрощение планирования. В идеале добавление процессоров к системе должно приводить к линейному росту ее производительности. Однако это не всегда так. Потери производительности могут возникать, например, при недостаточной пропускной способности шин из-за возрастания трафика между процессорами и основной памятью, а также между памятью и устройствами ввода/вывода. В действительности реальное увеличение производительности трудно оценить заранее, поскольку оно в значительной степени зависит от динамики поведения прикладных задач. Возможность масштабирования системы определяется не только архитектурой аппаратных средств, но зависит от заложенных свойств программного обеспечения. Масштабируемость программного обеспечения затрагивает все его уровни от простых механизмов передачи сообщений до работы с такими сложными объектами как мониторы транзакций и вся среда прикладной системы. В частности, программное обеспечение должно минимизировать трафик межпроцессорного обмена, который может препятствовать линейному росту производительности системы. Аппаратные средства (процессоры, шины и устройства ввода/вывода) являются только частью масштабируемой архитектуры, на которой программное обеспечение может обеспечить предсказуемый рост производительности. Важно понимать, что простой переход, например, на более мощный процессор может привести к перегрузке других компонентов системы. Это означает, что действительно масштабируемая система должна быть сбалансирована по всем параметрам.

**Совместимость и мобильность программного обеспечения**

Концепция программной совместимости впервые в широких масштабах была применена разработчиками системы IBM/360. Основная задача при проектировании всего ряда моделей этой системы заключалась в создании такой архитектуры, которая была бы одинаковой с точки зрения пользователя для всех моделей системы независимо от цены и производительности каждой из них. Огромные преимущества такого подхода, позволяющего сохранять существующий задел программного обеспечения при переходе на новые (как правило, более производительные) модели, были быстро оценены как производителями компьютеров, так и пользователями и, начиная с этого времени, практически все фирмы-поставщики компьютерного оборудования взяли на вооружение эти принципы, поставляя серии совместимых компьютеров. Следует заметить, однако, что со временем даже самая передовая архитектура неизбежно устаревает и возникает потребность внесения радикальных изменений архитектуру и способы организации вычислительных систем.

В настоящее время одним из наиболее важных факторов, определяющих современные тенденции в развитии информационных технологий, является ориентация компаний-поставщиков компьютерного оборудования на рынок прикладных программных средств. Это объясняется, прежде всего, тем, что для конечного пользователя, в конце концов, важно программное обеспечение, позволяющее решить его задачи, а не выбор той или иной аппаратной платформы. Переход от однородных сетей программно совместимых компьютеров к построению неоднородных сетей, включающих компьютеры разных фирм-производителей, в корне изменил и точку зрения на саму сеть: из сравнительно простого средства обмена информацией она превратилась в средство интеграции отдельных ресурсов — мощную распределенную вычислительную систему, каждый элемент которой (сервер или рабочая станция) лучше всего соответствует требованиям конкретной прикладной задачи. Этот переход выдвинул ряд новых требований.

Прежде всего, такая вычислительная среда должна позволять гибко менять количество и состав аппаратных средств и программного обеспечения в соответствии с меняющимися требованиями решаемых задач.

Во-вторых, она должна обеспечивать возможность запуска одних и тех же программных систем на различных аппаратных платформах, т.е. обеспечивать мобильность программного обеспечения.

В третьих, эта среда должна гарантировать возможность применения одних и тех же человеко-машинных интерфейсов на всех компьютерах, входящих в неоднородную сеть.

В условиях жесткой конкуренции производителей аппаратных платформ и программного обеспечения сформировалась концепция открытых систем, представляющая собой совокупность стандартов на различные компоненты вычислительной среды, предназначенных для обеспечения мобильности программных средств в рамках неоднородной, распределенной вычислительной системы. Одним из вариантов моделей открытой среды является модель OSE (Open System Environment), предложенная комитетом IEEE POSIX. На основе этой модели национальный институт стандартов и технологии США выпустил документ «Application Portability Profile (APP). The U.S. Government's Open System Environment Profile OSE/1 Version 2.0», который определяет рекомендуемые для федеральных учреждений США спецификации в области информационных технологий, обеспечивающие мобильность системного и прикладного программного обеспечения. Все ведущие производители компьютеров и программного обеспечения в США в настоящее время придерживаются требований этого документа.

**Надёжность и отказоустойчивость**

Одной из основных проблем построения вычислительных систем остаётся задача обеспечения их продолжительного функционирования. Важнейшей характеристикой вычислительных систем являетсянадежность. Повышение надежности основано на принципе предотвращения неисправностей путем снижения интенсивности отказов и сбоев за счет применения электронных схем и компонентов с высокой и сверхвысокой степенью интеграции, снижения уровня помех, облегченных режимов работы схем, обеспечение тепловых режимов их работы, а также за счет совершенствования методов сборки аппаратуры. Понятие надежности включает не только аппаратные средства, но и программное обеспечение. Главной целью повышения надежности систем является целостность хранимых в них данных. Единицей измерения надежности является среднее время наработки на отказ (MTBF — Mean Time Between Failure).

Отказоустойчивость — это такое свойство вычислительной системы, которое обеспечивает ей, как логической машине, возможность продолжения действий, заданных программой, после возникновения неисправностей. Введение отказоустойчивости требует избыточного аппаратного и программного обеспечения. Направления, связанные с предотвращением неисправностей и с отказоустойчивостью, — основные в проблеме надежности. Концепции параллельности и отказоустойчивости вычислительных систем естественным образом связаны между собой, поскольку в обоих случаях требуются дополнительные функциональные компоненты. Поэтому, собственно, на параллельных вычислительных системах достигается как наиболее высокая производительность, так и, во многих случаях, очень высокая надежность.

# 7. Операционные системы МВС. Коммуникации процессов.

**Операционные системы МВС**

Про ОС читайте это, всё копировать сюда я не собираюсь

<https://works.doklad.ru/view/t_Nad-J_iTQ/all.html>

**Коммуникация процессов**

Тут тоже муть, но хоть что-то, собранное в один текст

Взаимодействие процессов – основа для распараллеленного, эффективного решения задач с помощью группы процессов, координирующих свои действия друг с другом. В лекции рассмотрены некоторые классические схемы взаимодействия процессов при решении типовых задач (например, схема производитель – потребитель), а также виды взаимодействия процессов между собой с помощью передачи сообщений, сокетов, удаленных вызовов процедур и методов.

Рассмотрим теперь возможные механизмы для непосредственной коммуникации процессов и синхронизации их действий.

Наиболее распространенный из них - система сообщений; при этом процессы взаимодействуют между собой без обращений к общим переменным (сравните с алгоритмами производителя и потребителя раздела 9.4).

Средства коммуникации между процессами обеспечивают две операции вида:

* send (message ) – отправка сообщения message ; размер сообщения может быть постоянным или переменным;
* receive (message ) – получение сообщения в буфер message.

Если процессам P и Q требуется взаимодействовать между собой, им необходимо:

* Установить связь (communication link) друг с другом
* Обменяться сообщениями вида send/receive.

Реализация связи может быть физической (общая память, аппаратная шина) или логической (например, логические свойства).

При реализации коммуникационного механизма между процессами необходимо решить следующие вопросы:

* Как устанавливается связь?
* Можно ли установить связь более чем двух процессов?
* Сколько связей может быть установлено между двумя заданными процессами?
* Какова пропускная способность линии связи?
* Является ли длина сообщения по линии связи постоянной или переменной?
* Является ли связь ненаправленной или двунаправленной (дуплексной)?

**Непосредственная коммуникация**

При непосредственной коммуникации (direct communication) процессы именуют друг друга явно – по именам или по адресам (указателям), которые указываются в вызовах коммуникационных примитивов, например:

* send (P, message ) – послать сообщение процессу P ;
* receive (Q, message ) – получить сообщение от процесса Q.

При данном способе коммуникации свойства линии связи следующие:

* Связь устанавливается автоматически (ее реализуют операционная система или отдельные коммуникационные инструменты).
* Связь ассоциируется только с одной парой взаимодействующих процессов.
* Между каждой парой процессов всегда только одна связь.
* Связь может быть ненаправленной, но, как правило, она двунаправленная (процесс может получить сообщение от другого явно заданного процесса и принять от него сообщение).

**Косвенная коммуникация процессов**

При косвенной коммуникации (indirect communication) сообщения направляются и получаются через почтовые ящики ( mailboxes ), или порты ( ports ) – системные структуры, предназначенные для приема, хранения и передачи сообщений. Для определенности будем использовать термин почтовый ящик.

Каждый почтовый ящик имеет уникальный идентификатор.

Процессы могут взаимодействовать, только если они имеют общий почтовый ящик.

Свойства линии связи в этом случае следующие:

* Связь устанавливается, только если процессы имеют общий почтовый ящик.
* Связь одного процесса может быть установлена со многими процессами (которым доступен тот же почтовый ящик).
* Каждая пара процессов может иметь несколько линий связи (так как сообщения могут посылаться через различные почтовые ящики).
* Связь может быть ненаправленной или двунаправленной.

При косвенном способе коммуникации процессы используют набор операций вида:

* Создать новый почтовый ящик.
* Отправить (принять) сообщение через почтовый ящик.
* Удалить почтовый ящик

Основные операции коммуникации принимают вид:

* send (A, message ) – послать сообщение в почтовый ящик A.
* receive (A, message ) – получить сообщение из почтового ящика A.

# 8. Функции ОС. Процесс. Поток. Синхронизация параллельных процессов.

**Основные Функции**

Для решения любой задачи на компьютере необходимы, по крайней мере, два вида ресурсов: оперативная память для хранения программы и данных и процессор для исполнения команд. Указанные ресурсы могут быть предоставлены задаче самим пользователем, если он вручную разместит в основной памяти программу и данные и введет в машину информацию для запуска процессора. Однако такой способ не приемлем для больших программ, т.к. является очень трудоемким и медленным. Дело в том, что элементарные операции при работе с устройствами компьютера и по управлению его ресурсами - это операции очень низкого уровня, состоящие из нескольких сотен и тысяч элементарных команд.

Операционная система освобождает пользователя от долгой и кропотливой работы, связанной с распределением ресурсов компьютера, управлением устройствами, организацией выполнения программ, выполняя эти действия автоматически.

Основными функциями ОС являются следующие:

* запуск программ и контроль за их прохождением;
* управление оперативной памятью;
* управление устройствами ввода и вывода;
* управление внешней памятью;
* управление взаимодействием одновременно работающих задач;
* обработка вводимых команд для обеспечения взаимодействия с пользователем.

Операционная система обычно состоит из управляющей части и набора системных программ (обслуживающая часть).

Управляющая часть содержится в нескольких файлах. Ее функциями являются: распределение вычислительных ресурсов, запуск и контроль выполнения программ, управление стандартными внешними устройствами, управление файлами. Для обеспечения работы с дополнительными внешними устройствами в состав управляющей части операционной системы входят драйверы. Это очень небольшие программы, которые позволяют работать с конкретными внешними устройствами. Наличие драйверов позволяет подключать к компьютеру различные типы внешних устройств, причем для этого не нужно коренным образом перестраивать вычислительную среду, а достаточно включить в состав ОС определенный драйвер.

В набор системных программ входят программы, также поставляемые в виде отдельных файлов. Они выполняют действия обслуживающего характера, расширяющие возможности ядра операционной системы, предоставляющие дополнительные возможности и удобства пользователю.

Для нормальной работы компьютера определенная часть операционной системы, называемая резидентной, должна постоянно находиться в основной памяти, сокращая, таким образом, объем памяти, доступный для прикладных программ. Другие части системы автоматически загружаются в память из внешних устройств по мере необходимости. После выполнения требуемых действий занимаемые ими области памяти освобождаются.

Существующие операционные системы принято классифицировать следующим образом. По числу одновременно обслуживаемых рабочих мест ОС разделяются на однопользовательские и сетевые.

По количеству одновременно выполняемых программ выделяют однозадачные и многозадачные ОС. В однозадачном режиме все ресурсы компьютера предоставляются только одной программе, которая выполняет обработку данных. При работе в многозадачном (мультипрограммном) режиме несколько не зависимых друг от друга программ выполняют обработку данных одновременно, т.е. параллельно. При этом программы делят ресурсы компьютера между собой.

Про процессы и потоки читаем здесь

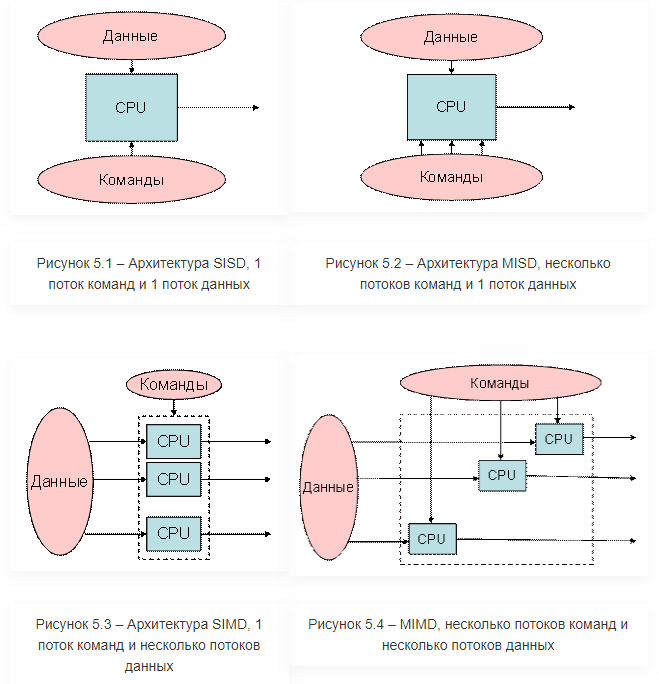
https://moodle.kstu.ru/mod/page/view.php?id=49

# 9. Классификация архитектур по параллельной обработке данных (SISD, MISD, SIMD, MIMD).

В 1966 г. М. Флинном (Flynn) был предложен чрезвычайно удобный подход к классификации архитектур вычислительных систем. В его основу было положено понятие потока, под которым понимается последовательность элементов, команд или данных, обрабатываемая процессором.

Соответствующая система классификации основана на рассмотрении числа потоков инструкций и потоков данных и описывает четыре архитектурных класса:

* SISD (Single Instruction Single Data) – 1 поток команд и 1 поток данных.
* MISD (Multiple Instruction Single Data) – несколько потоков команд и 1 поток данных.
* SIMD (Single Instruction Multiple Data) – 1 поток команд и несколько потоков данных.
* MIMD (Multiple Instruction Multiple Data) – несколько потоков команд и несколько потоков данных.



**SISD (single instruction stream / single data stream)** – одиночный поток команд и одиночный поток данных. К этому классу относятся последовательные компьютерные системы, которые имеют один центральный процессор, способный обрабатывать только один поток последовательно исполняемых инструкций.

В настоящее время практически все высокопроизводительные системы имеют более одного центрального процессора, однако каждый из них выполняет несвязанные потоки инструкций, что делает такие системы комплексами SISD-систем, действующих на разных пространствах данных.

Для увеличения скорости обработки команд и скорости выполнения арифметических операций может применяться конвейерная обработка. В случае векторных систем векторный поток данных следует рассматривать как поток из одиночных неделимых векторов.

Примерами компьютеров с архитектурой SISD могут служить большинство рабочих станций Compaq, Hewlett-Packard и Sun Microsystems.

**MISD (multiple instruction stream / single data stream)** – множественный поток команд и одиночный поток данных. Теоретически в этом типе машин множество инструкций должно выполняться над единственным потоком данных. До сих пор ни одной реальной машины, попадающей в данный класс, создано не было. В качестве аналога работы такой системы, по-видимому, можно рассматривать работу банка. С любого терминала можно подать команду и что-то сделать с имеющимся банком данных. Поскольку база данных одна, а команд много, мы имеем дело с множественным потоком команд и одиночным потоком данных.

**SIMD (single instruction stream / multiple data stream)** – одиночный поток команд и множественный поток данных. Эти системы обычно имеют большое количество процессоров (от 1024 до 16384), которые могут выполнять одну и ту же инструкцию относительно разных данных в жесткой конфигурации. Единственная инструкция параллельно выполняется над многими элементами данных.

Примерами SIMD-машин являются системы CPP DAP, Gamma II и Quadrics Apemille.

Другим подклассом SIMD-систем являются векторные компьютеры. Векторные компьютеры манипулируют массивами сходных данных подобно тому, как скалярные машины обрабатывают отдельные элементы таких массивов. Это делается за счет использования специально сконструированных векторных центральных процессоров. Когда данные обрабатываются посредством векторных модулей, результаты могут быть выданы на один, два или три такта частотогенератора (такт частотогенератора является основным временным параметром системы). При работе в векторном режиме векторные процессоры обрабатывают данные практически параллельно, что делает их в несколько раз более быстрыми, чем при работе в скалярном режиме. Примерами систем подобного типа являются, например, компьютеры Hitachi S3600.

**MIMD (multiple instruction stream / multiple data stream)** – множественный поток команд и множественный поток данных. Эти машины параллельно выполняют несколько потоков инструкций над различными потоками данных. В отличие от упомянутых выше многопроцессорных SISD-машин, команды и данные связаны, потому что они представляют различные части одной и той же задачи. Например, MIMD-системы могут параллельно выполнять множество подзадач с целью сокращения времени выполнения основной задачи.

Большое разнообразие попадающих в MIMD класс вычислительных систем делает классификацию Флинна не полностью адекватной. Действительно, и четырехпроцессорный SX-5 компании NEC, и тысячепроцессорный Cray T3E попадают в этот класс. Это заставляет использовать другой подход к классификации, иначе описывающий классы компьютерных систем.

# 10. Распределённые операционные системы. Классификация мультипроцессорных операционных систем.

**Немного про распределённые ОС**

Распределенная ОС, динамически и автоматически распределяя работы по различным машинам системы для обработки, заставляет набор сетевых машин работать как виртуальный унипроцессор. Пользователь распределенной ОС, вообще говоря, не имеет сведений о том, на какой машине выполняется его работа.

Распределенная ОС существует как единая операционная система в масштабах вычислительной системы. Каждый компьютер сети, работающей под управлением распределенной ОС, выполняет часть функций этой глобальной ОС. Распределенная ОС объединяет все компьютеры сети в том смысле, что они работают в тесной кооперации друг с другом для эффективного использования всех ресурсов компьютерной сети. Два значения термина «сетевая ОС» В настоящее время практически все сетевые операционные системы еще очень далеки от идеала истинной распределенности. Степень автономности каждого компьютера в сети, работающей под управлением сетевой операционной системы, значительно выше по сравнению с компьютерами, работающими под управлением распределенной ОС.

В результате сетевая ОС может рассматриваться как набор операционных систем отдельных компьютеров, составляющих сеть. На разных компьютерах сети могут выполняться одинаковые или разные ОС. Например, на всех компьютерах сети может работать одна и та же ОС UNIX. Более реалистичным вариантом является сеть, в которой работают разные ОС, например часть компьютеров работает под управлением UNIX, часть — под управлением NetWare, а остальные — под управлением Windows NT и Windows 98. Все эти операционные системы функционируют независимо друг от друга в том смысле, что каждая из них принимает независимые решения о создании и завершении своих собственных процессов и управлении локальными ресурсами. Но в любом случае операционные системы компьютеров, работающих в сети, должны включать взаимно согласованный набор коммуникационных протоколов для организации взаимодействия процессов, выполняющихся на разных компьютерах сети, и разделения ресурсов этих компьютеров между пользователями сети.

Если операционная система отдельного компьютера позволяет ему работать в сети, то есть предоставлять свои ресурсы в общее пользование и/или потреблять ресурсы других компьютеров сети, то такая операционная система отдельного компьютера также называется сетевой ОС.

Таким образом, термин «сетевая операционная система» используется в двух значениях: во-первых, как совокупность ОС всех компьютеров сети и, во-вторых, как операционная система отдельного компьютера, способного работать в сети. Исходя из этого определения следует, что такие операционные системы, как, например, Windows NT, NetWare, Solans, HP-UX, являются сетевыми, поскольку все они обладают средствами, которые позволяют их пользователям работать в сети. Функциональные компоненты сетевой ОС.

В настоящее время развитие высокопроизводительных вычислительных систем идет по четырем основным направлениям: векторно-конвейерные суперкомпьютеры (PVP), SMP системы, MPP системы и кластерные системы. Рассмотрим основные особенности перечисленных архитектур.

Подробнее про эти системы смотрите в следующих вопросах.

# 11. Идея распараллеливания вычислений. Типы параллелизма.

Слизано с вики, там есть ссылки на подробные статьи, если интересно (нет): <https://ru.wikipedia.org/wiki/Параллельные_вычислительные_системы>

Идея распараллеливания вычислений основана на том, что большинство задач может быть разделено на набор меньших задач, которые могут быть решены одновременно. Обычно параллельные вычисления требуют координации действий. Параллельные вычисления существуют в нескольких формах: параллелизм на уровне битов, параллелизм на уровне инструкций, параллелизм данных, параллелизм задач. Параллельные вычисления использовались много лет в основном в высокопроизводительных вычислениях, но в последнее время к ним возрос интерес вследствие существования физических ограничений на рост тактовой частоты процессоров. Параллельные вычисления стали доминирующей парадигмой в архитектуре компьютеров, в основном в форме многоядерных процессоров.

**Параллелизм на уровне битов**

Эта форма параллелизма основана на увеличении размера машинного слова. Увеличение размера машинного слова уменьшает количество операций, необходимых процессору для выполнения действий над переменными, чей размер превышает размер машинного слова. К примеру: на 8-битном процессоре нужно сложить два 16-битных целых числа. Для этого вначале нужно сложить нижние 8 бит чисел, затем сложить верхние 8 бит и к результату их сложения прибавить значение флага переноса. Итого 3 инструкции. С 16-битным процессором можно выполнить эту операцию одной инструкцией.

Исторически 4-битные микропроцессоры были заменены 8-битными, затем появились 16-битные и 32-битные. 32-битные процессоры долгое время были стандартом в повседневных вычислениях. С появлением технологии x86-64 для этих целей стали использовать 64-битные процессоры.

**Параллелизм на уровне инструкций**

Компьютерная программа — это, по существу, поток инструкций, выполняемых процессором. Но можно изменить порядок этих инструкций, распределить их по группам, которые будут выполняться параллельно, без изменения результата работы всей программы. Данный приём известен как параллелизм на уровне инструкций. Продвижения в развитии параллелизма на уровне инструкций в архитектуре компьютеров происходили с середины 1980-х до середины 1990-х.

Современные процессоры имеют многоступенчатый конвейер команд. Каждой ступени конвейера соответствует определённое действие, выполняемое процессором в этой инструкции на этом этапе. Процессор с N ступенями конвейера может иметь одновременно до N различных инструкций на разном уровне законченности. Классический пример процессора с конвейером — это RISC-процессор с 5-ю ступенями: выборка инструкции из памяти (IF), декодирование инструкции (ID), выполнение инструкции (EX), доступ к памяти (MEM), запись результата в регистры (WB). Процессор Pentium 4 имеет конвейер в 31 ступень.

Некоторые процессоры, дополнительно к использованию конвейеров, обладают возможностью выполнять несколько инструкций одновременно, что даёт дополнительный параллелизм на уровне инструкций. Возможна реализация данного метода при помощи суперскалярности, когда инструкции могут быть сгруппированы вместе для параллельного выполнения (если в них нет зависимости между данными (завимости по данным)). Также возможны реализации с использованием явного параллелизма на уровне инструкций: VLIW и EPIC.

**Параллелизм данных**

Основная идея подхода, основанного на параллелизме данных, заключается в том, что одна операция выполняется сразу над всеми элементами массива данных. Различные фрагменты такого массива обрабатываются на векторном процессоре или на разных процессорах параллельной машины. Распределением данных между процессорами занимается программа. Векторизация или распараллеливание в этом случае чаще всего выполняется уже на этапе компиляции — перевода исходного текста программы в машинные команды. Роль программиста в этом случае обычно сводится к заданию настроек векторной или параллельной оптимизации компилятору, директив параллельной компиляции, использованию специализированных языков для параллельных вычислений.

**Параллелизм задач**

Стиль программирования, основанный на параллелизме задач, подразумевает, что вычислительная задача разбивается на несколько относительно самостоятельных подзадач и каждый процессор загружается своей собственной подзадачей.

# 12. SMP-архитектура.

SMP (symmetric multiprocessing) – симметричная многопроцессорная архитектура. Главной особенностью систем с архитектурой SMP является наличие общей физической памяти, разделяемой всеми процессорами.



Память служит, в частности, для передачи сообщений между процессорами, при этом все вычислительные устройства при обращении к ней имеют равные права и одну и ту же адресацию для всех ячеек памяти. Поэтому SMP-архитектура называется симметричной. Последнее обстоятельство позволяет очень эффективно обмениваться данными с другими вычислительными устройствами. SMP-система строится на основе высокоскоростной системной шины (SGI PowerPath, Sun Gigaplane, DEC TurboLaser), к слотам которой подключаются функциональные блоки типов: процессоры (ЦП), подсистема ввода/вывода (I/O) и т. п. Для подсоединения к модулям I/O используются уже более медленные шины (PCI, VME64). Наиболее известными SMP-системами являются SMP-cерверы и рабочие станции на базе процессоров Intel (IBM, HP, Compaq, Dell, ALR, Unisys, DG, Fujitsu и др.) Вся система работает под управлением единой ОС (обычно UNIX-подобной, но для Intel-платформ поддерживается Windows NT). ОС автоматически (в процессе работы) распределяет процессы по процессорам, но иногда возможна и явная привязка.

Основные преимущества SMP-систем:

* простота и универсальность для программирования. Архитектура SMP не накладывает ограничений на модель программирования, используемую при создании приложения: обычно используется модель параллельных ветвей, когда все процессоры работают независимо друг от друга. Однако можно реализовать и модели, использующие межпроцессорный обмен. Использование общей памяти увеличивает скорость такого обмена, пользователь также имеет доступ сразу ко всему объему памяти. Для SMP-систем существуют довольно эффективные средства автоматического распараллеливания;
* простота эксплуатации. Как правило, SMP-системы используют систему кондиционирования, основанную на воздушном охлаждении, что облегчает их техническое обслуживание;
* относительно невысокая цена.

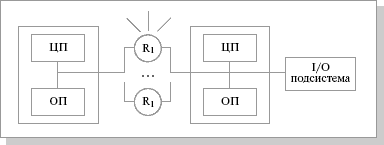
Недостатки:

* системы с общей памятью плохо масштабируются.

Этот существенный недостаток SMP-систем не позволяет считать их по-настоящему перспективными. Причиной плохой масштабируемости является то, что в данный момент шина способна обрабатывать только одну транзакцию, вследствие чего возникают проблемы разрешения конфликтов при одновременном обращении нескольких процессоров к одним и тем же областям общей физической памяти. Вычислительные элементы начинают друг другу мешать. Когда произойдет такой конфликт, зависит от скорости связи и от количества вычислительных элементов. В настоящее время конфликты могут происходить при наличии 8-24 процессоров. Кроме того, системная шина имеет ограниченную (хоть и высокую) пропускную способность (ПС) и ограниченное число слотов. Все это очевидно препятствует увеличению производительности при увеличении числа процессоров и числа подключаемых пользователей. В реальных системах можно задействовать не более 32 процессоров. Для построения масштабируемых систем на базе SMP используются кластерные или NUMA-архитектуры. При работе с SMP-системами используют так называемую парадигму программирования с разделяемой памятью (shared memory paradigm).

# 13. MPP-архитектура.

MPP (massive parallel processing) – массивно-параллельная архитектура. Главная особенность такой архитектуры состоит в том, что память физически разделена. В этом случае система строится из отдельных модулей, содержащих процессор, локальный банк операционной памяти (ОП), коммуникационные процессоры (рутеры) или сетевые адаптеры, иногда – жесткие диски и/или другие устройства ввода/вывода. По сути, такие модули представляют собой полнофункциональные компьютеры (см. рис.3.2). Доступ к банку ОП из данного модуля имеют только процессоры (ЦП) из этого же модуля. Модули соединяются специальными коммуникационными каналами. Пользователь может определить логический номер процессора, к которому он подключен, и организовать обмен сообщениями с другими процессорами. Используются два варианта работы операционной системы (ОС) на машинах MPP-архитектуры. В одном полноценная операционная система (ОС) работает только на управляющей машине (front-end), на каждом отдельном модуле функционирует сильно урезанный вариант ОС, обеспечивающий работу только расположенной в нем ветви параллельного приложения. Во втором варианте на каждом модуле работает полноценная UNIX-подобная ОС, устанавливаемая отдельно.



Главным преимуществом систем с раздельной памятью является хорошая масштабируемость: в отличие от SMP-систем, в машинах с раздельной памятью каждый процессор имеет доступ только к своей локальной памяти, в связи с чем не возникает необходимости в потактовой синхронизации процессоров. Практически все рекорды по производительности на сегодня устанавливаются на машинах именно такой архитектуры, состоящих из нескольких тысяч процессоров (ASCI Red, ASCI Blue Pacific).

Недостатки:

* отсутствие общей памяти заметно снижает скорость межпроцессорного обмена, поскольку нет общей среды для хранения данных, предназначенных для обмена между процессорами. Требуется специальная техника программирования для реализации обмена сообщениями между процессорами;
* каждый процессор может использовать только ограниченный объем локального банка памяти;
* вследствие указанных архитектурных недостатков требуются значительные усилия для того, чтобы максимально использовать системные ресурсы. Именно этим определяется высокая цена программного обеспечения для массивно-параллельных систем с раздельной памятью.

Системами с раздельной памятью являются суперкомпьютеры МВС-1000, IBM RS/6000 SP, SGI/CRAY T3E, системы ASCI, Hitachi SR8000, системы Parsytec.

# 14. PVP-архитектура.

Основным признаком PVP-систем является наличие специальных векторно-конвейерных процессоров, в которых предусмотрены команды однотипной обработки векторов независимых данных, эффективно выполняющиеся на конвейерных функциональных устройствах. Как правило, несколько таких процессоров (1-16) работают одновременно с общей памятью (аналогично SMP) в рамках многопроцессорных конфигураций. Несколько узлов могут быть объединены с помощью коммутатора (аналогично MPP). Поскольку передача данных в векторном формате осуществляется намного быстрее, чем в скалярном (максимальная скорость может составлять 64 Гбайт/с, что на 2 порядка быстрее, чем в скалярных машинах), то проблема взаимодействия между потоками данных при распараллеливании становится несущественной. И то, что плохо распараллеливается на скалярных машинах, хорошо распараллеливается на векторных. Таким образом, системы PVP-архитектуры могут являться машинами общего назначения (general purpose systems). Однако, поскольку векторные процессоры весьма дорого стоят, эти машины не могут быть общедоступными.

Парадигма программирования на PVP-системах предусматривает векторизацию циклов (для достижения разумной производительности одного процессора) и их распараллеливание (для одновременной загрузки нескольких процессоров одним приложением).

На практике рекомендуется выполнять следующие процедуры:

* производить векторизацию вручную, чтобы перевести задачу в матричную форму. При этом, в соответствии с длиной вектора, размеры матрицы должны быть кратны 128 или 256;
* работать с векторами в виртуальном пространстве, разлагая искомую функцию в ряд и оставляя число членов ряда, кратное 128 или 256.

За счет большой физической памяти (доли терабайта) даже плохо векторизуемые задачи на PVP-системах решаются быстрее на машинах со скалярными процессорами.

# 15. Составные коммутаторы. Коммутатор Клоза. Баньян-сети. Распределённые составные коммутаторы.

Коммутаторы обеспечивают параллельную работу процессорных модулей и вычислительных машин в системах. Сетевые коммутаторы разделяются на простые сетевые коммутаторы и составные сетевые коммутаторы. Простые коммутаторы имеют малую задержку, но бывают использованы только для построения систем с малым числом узлов. Составные коммутаторы строятся путем объединения простых коммутаторов.

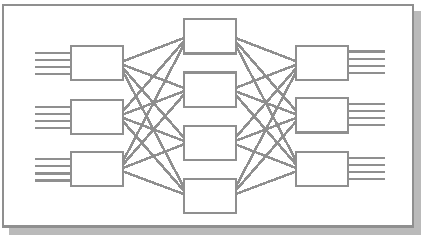
Простые коммутаторы имеют ограничения на число входов и выходов, а также могут требовать большого количества оборудования при увеличении этого числа (в случае пространственных коммутаторов). Поэтому для построения коммутаторов с большим количеством входов и выходов используют совокупность простых коммутаторов, объединенных с помощью линий "точка-точка".

Составные коммутаторы имеют задержку, пропорциональную количеству простых коммутаторов, через которые проходит сигнал от входа до выхода, т.е. числу каскадов. Однако объем оборудования составного коммутатора меньше, чем простого с тем же количеством входов и выходов.

Чаще всего составные коммутаторы строятся из прямоугольных коммутаторов 2 х 2 с двумя входами и выходами. Они имеют два состояния: прямое пропускание входов на соответствующие выходы и перекрестное пропускание. Коммутатор 2 х 2 состоит из собственно блока коммутации данных и блока управления. Блок управления в зависимости от поступающих на него управляющих сигналов определяет, какой тип соединения следует осуществить в блоке коммутации - прямой или перекрестный. При этом если оба входа хотят соединиться с одним выходом, то коммутатор разрешает конфликт и связывает с данным выходом только один вход, а запрос на соединение со стороны второго блокируется или отвергается.

**Коммутатор Клоза**

Коммутатор Клоза может быть построен в качестве альтернативы для прямоугольного коммутатора с (m x d) входами и (m x d) выходами. Он формируется из трех каскадов коммутаторов: m коммутаторов (d x d) во входном каскаде, m коммутаторов (d x d) в выходном и d промежуточных коммутаторов (m x m).



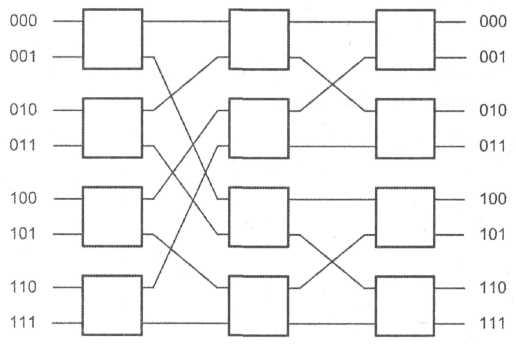
Соединения внутри коммутатора устроены следующим образом:

* j-й выход i-ого коммутатора входного каскада соединен с i-ым входом j-ого промежуточного коммутатора;
* j-й вход k-ого коммутатора выходного каскада соединен с k-ым выходом j-ого промежуточного коммутатора.

Данный тип составных коммутаторов позволяет соединять любой вход с любым выходом, однако при установленных соединениях добавление нового соединения может потребовать разрыва и переустановления всех соединений.

**Баньян-сети**

Коммутаторы этого типа строятся на базе прямоугольных коммутаторов таким образом, что существует только один путь от каждого входа к каждому выходу.



Наиболее важной разновидностью баньян-сетей является дельта-сеть. Она формируется из прямоугольных коммутаторов (a x b) и представляет собой n-каскадный коммутатор с an входами и bn выходами. Составляющие коммутаторы соединены так, что для соединения любого входа и выхода образуется единственный путь одинаковой для всех пар входов и выходов длины.

# 16. Простые коммутаторы. Шинная архитектура. Разрешение конфликтов. Алгоритмы арбитража.

Коммуникационные среды вычислительных систем (ВС) состоят из адаптеров вычислительных модулей (ВМ) и коммутаторов, обеспечивающих соединения между ними. Используются как простые коммутаторы, так и составные, компонуемые из набора простых. Простые коммутаторы могут соединять лишь малое число ВМ в силу физических ограничений, однако обеспечивают при этом минимальную задержку при установлении соединения. Составные коммутаторы, обычно строящиеся из простых в виде многокаскадных схем с помощью линий "точка-точка", преодолевают ограничение на малое количество соединений, однако увеличивают и задержки.

**Простые коммутаторы**

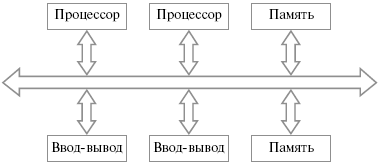
Типы простых коммутаторов:

* с временным разделением ;
* с пространственным разделением.

Достоинства: простота управления и высокое быстродействие. Недостатки: малое количество входов и выходов.

**Шины**

Простые коммутаторы с временным разделением называются также шинами или шинными структурами. Все устройства подключаются к общей информационной магистрали, используемой для передачи информации между ними (рис. 9.1). Обычно шина является пассивным элементом, управление передачами осуществляется передающими и принимающими устройствами.



Передающее устройство сначала получает доступ к шине, далее пытается установить контакт с устройством-адресатом и определить его способность к приему данных. Принимающее устройство распознает свой адрес на шине и отвечает на запрос передающего. Далее передающее устройство сообщает, какие действия должно произвести принимающее устройство в ходе взаимодействия. После этого происходит передача данных.

Так как шина является общим ресурсом, за доступ к которому соревнуются подключенные к ней устройства, необходимы методы управления предоставлением доступа устройств к шине. Возможно использование центрального устройства для управления доступом к шине, однако это уменьшает масштабируемость и гибкость системы.

Для разрешения конфликтов, возникающих при одновременном запросе устройств на доступ к шине, используются различные приемы, в частности:

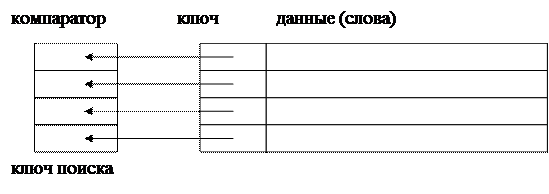
* назначение каждому устройству уникального приоритета (статического или динамического);
* использование очереди запросов FIFO;
* выделение фиксированных временных интервалов каждому устройству.

# 17. Ассоциативные процессоры.

Ассоциативные системы относятся к классу «один поток команд – множество потоков данных» (SIMD).

Ассоциативный способ обработки данных позволяет преодолеть многие ограничения, присущие адресному доступу к памяти за счет специального способа работы (безадресного).

Ассоциативная память представляет собой хранилище данных, в котором обращение к элементам (словам) происходит по полю ключа, хранящегося вместе с данными. Схема сравнения (компаратор) выполняет побитовое сравнение входного ключа со значениями ключей в словах ассоциативной памяти. В результате оказываются выбранными те слова памяти, которые имеют аналогичный ключ.



Для выполнения операции поиска свободных слов и поиска по некоторым битам поля ключа в устройство управления ассоциативной памяти включается регистр маски, биты которого указывают, какие биты регистра ключа (регистра ассоциативных признаков) должны сравниваться компаратором с битами ключей слов ассоциативной памяти.

Идеи ассоциативной памяти были использованы в процессоре STARAN. Это матричный процессор, способный выполнять ассоциативный поиск с параллельным сравнением разрядов и последовательным сравнением слов и наоборот, т.е. с параллельным сравнением слов и последовательным сравнением разрядов по отношению к 256 процессорам, объединенных в единую матрицу.

# 18. Непредсказуемость производительности. Аспекты масштабирования. Совместимость и мобильность программного обеспечения. Недолговечность архитектур.

Тут я не нашёл адекватных материалов, воображение вам в руки

# 19. Кластерные технологии. Архитектуры кластеров.

Кластер — группа компьютеров, объединённых высокоскоростными каналами связи, представляющая с точки зрения пользователя единый аппаратный ресурс. Кластер - слабо связанная совокупность нескольких вычислительных систем, работающих совместно для выполнения общих приложений, и представляющихся пользователю единой системой. Один из первых архитекторов кластерной технологии Грегори Пфистер дал кластеру следующее определение: Кластер — это разновидность параллельной или распределённой системы, которая:

* состоит из нескольких связанных между собой компьютеров;
* используется как единый, унифицированный компьютерный ресурс.

Обычно различают следующие основные виды кластеров:

отказоустойчивые кластеры (High-availability clusters, HA, кластеры высокой доступности)

* кластеры с балансировкой нагрузки (Load balancing clusters)
* вычислительные кластеры (High performance computing clusters, HPC)
* системы распределенных вычислений

В большинстве случаев, кластеры серверов функционируют на раздельных компьютерах. Это позволяет повышать производительность за счёт распределения нагрузки на аппаратные ресурсы и обеспечивает отказоустойчивость на аппаратном уровне.

Однако, принцип организации кластера серверов (на уровне программного протокола) позволяет исполнять по нескольку программных серверов на одном аппаратном. Такое использование может быть востребовано:

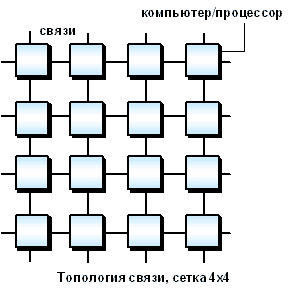
* при разработке и тестировании кластерных решений;
* при необходимости обеспечить доступность кластера только с учётом частых изменений конфигурации серверов — членов кластера, требующих их перезагрузки (перезагрузка производится поочерёдно) в условиях ограниченных аппаратных ресурсов.

**Архитектуры**

Архитектура кластерной системы (способ соединения процессоров друг с другом) в большей степени определяет ее производительность, чем тип используемых в ней процессоров. Критическим параметром, влияющим на величину производительности такой системы, является расстояние между процессорами. Так, соединив вместе 10 персональных компьютеров, мы получим систему для проведения высокопроизводительных вычислений, проблема, однако, будет состоять в нахождении наиболее эффективного способа соединения стандартных средств друг с другом, поскольку при увеличении производительности каждого процессора в 10 раз производительность системы в целом в 10 раз не увеличится.

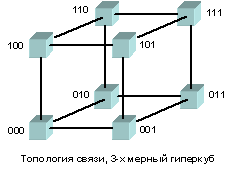
Рассмотрим для примера задачу построения симметричной 16-ти процессорной системы, в которой все процессоры были бы равноправны. Наиболее естественным представляется соединение в виде плоской решетки, где внешние концы используются для подсоединения внешних устройств.

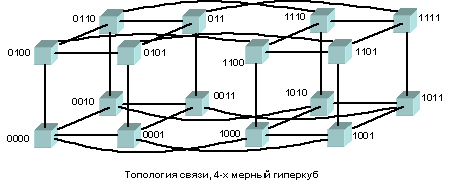
*Схема соединения процессоров в виде плоской решетки*



При таком типе соединения максимальное расстояние между процессорами окажется равным 6 (количество связей между процессорами, отделяющих самый ближний процессор от самого дальнего). Теория же показывает, что если в системе максимальное расстояние между процессорами больше 4, то такая система не может работать эффективно. Поэтому, при соединении 16 процессоров друг с другом плоская схема является не эффективной. Для получения более компактной конфигурации необходимо решить задачу о нахождении фигуры, имеющей максимальный объем при минимальной площади поверхности. В трехмерном пространстве таким свойством обладает шар. Но поскольку нам необходимо построить узловую систему, то вместо шара приходится использовать куб (если число процессоров равно 8) или гиперкуб, если число процессоров больше 8. Размерность гиперкуба будет определяться в зависимости от числа процессоров, которые необходимо соединить. Так, для соединения 16 процессоров потребуется 4-х мерный гиперкуб. Для его построения следует взять обычный 3-х мерный куб, сдвинуть в еще одном направлении и, соединив вершины, получить гиперкуб размером 4.

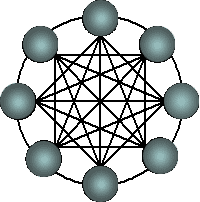
*Примеры гиперкубов*





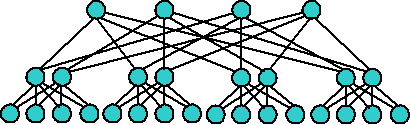
Архитектура гиперкуба является второй по эффективности, но самой наглядной. Используются и другие топологии сетей связи: трехмерный тор, "кольцо", "звезда" и другие.

*Архитектура кольца с полной связью по хордам (Chordal Ring)*

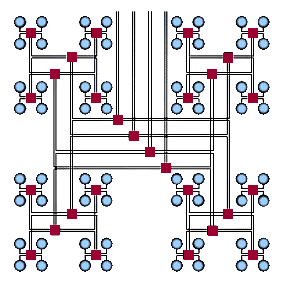


Наиболее эффективной является архитектура с топологией "толстого дерева" (fat-tree). Архитектура "fat-tree" (hypertree) предложена Лейзерсоном (Charles E. Leiserson) в 1985 году. Процессоры локализованы в листьях дерева, в то время как внутренние узлы дерева скомпонованы во внутреннюю сеть. Поддеревья могут общаться между собой, не затрагивая более высоких уровней сети.

*Кластерная архитектура "Fat-tree"*



*Кластерная архитектура "Fat-tree" (вид сверху на предыдущую схему)*



Поскольку способ соединения процессоров друг с другом больше влияет на производительность кластера, чем тип используемых в ней процессоров, то может оказаться более рентабельным создать систему из большего числа дешевых компьютеров, чем из меньшего числа дорогих. В кластерах, как правило, используются операционные системы, стандартные для рабочих станций, чаще всего, свободно распространяемые - Linux,FreeBSD, вместе со специальными средствами поддержки параллельного программирования и балансировки нагрузки. При работе с кластерами, также как и с MPP системами, используют так называемую Massive Passing Programming Paradigm - парадигму программирования с передачей данных (чаще всего - MPI). Дешевизна подобных систем оборачивается большими накладными расходами на взаимодействие параллельных процессов между собой, что сильно сужает потенциальный класс решаемых задач.

# 20. ГРИД. Типы ГРИД-систем.

Грид (англ. grid — решётка, сеть) — это система, которая, во-первых, распределяет ресурсы, не находящиеся под единым центром управления, во-вторых, использует общие протоколы и интерфейсы, в-третьих, обеспечивает нужный уровень обслуживания[Источник 1]. Грид - это система, которая координирует распределенные ресурсы посредством стандартных, открытых, универсальных протоколов и интерфейсов для обеспечения нетривиального качества обслуживания.

**История**

Термин «облачные вычисления» появился в начале 1990-х гг. как метафора о такой же лёгкости доступа к вычислительным ресурсам, как и к электрической сети (англ. power grid) в сборнике под редакцией Яна Фостера и Карла Кессельмана «The Grid: Blueprint for a new computing infrastructure».

Идеи Grid-системы (включая идеи из областей распределённых вычислений, объектно-ориентированного программирования, использования компьютерных кластеров, веб-сервисов и др.) были собраны и объединены Иэном Фостером, Карлом Кессельманом и Стивом Тики, которых часто называют отцами технологии ГРИД. Они начали создание набора инструментов для Grid Globus Toolkit, который включает не только инструменты менеджмента вычислений, но и инструменты управления ресурсами хранения данных, обеспечения безопасности доступа к данным и к самому гриду, мониторинга использования и передвижения данных, а также инструментарий для разработки дополнительных сервисов Грид. В настоящее время этот набор инструментария является де факто стандартом для построения инфраструктуры на базе технологии грид, хотя на рынке существует множество других инструментариев для Grid-системы как в масштабе предприятия, так и в глобальном.

**Задача «грид»**

Основная задача «грид» - согласованное распределение ресурсов:[1] и решение задач в условиях динамических, многопрофильных виртуальных организаций.

Исследования и разработки в сообществе грид привели к разработке протоколов, сервисов и инструментария, направленного именно на те проблемы, которые возникают при попытке создания масштабируемых ВО. Эти технологии включают в себя: решения по безопасности, поддерживающие управление сертификацией и политиками безопасности, когда вычисления производятся несколькими организациями;

* Протоколы управления ресурсами и сервисами, поддерживающие безопасный удаленный доступ к вычислительным ресурсам и ресурсам данных, а также перераспределение различных ресурсов;
* Протоколы запроса информации и сервисы, обеспечивающие настройку и мониторинг состояния ресурсов, организаций и сервисов; <
* Сервисы обработки данных, обеспечивающие поиск и передачу наборов данных между системами хранения данных и приложениями.

**Концепция**

Грид-технологии (Grid) позволяют создать географически распределенные вычислительные инфраструктуры, которые объединяют разнородные ресурсы и реализуют возможность коллективного доступа к этим ресурсам. Принципиальной новизной этих технологий является объединение ресурсов путем создания компьютерной инфраструктуры нового типа, обеспечивающей глобальную интеграцию информационных и вычислительных ресурсов на основе сетевых технологий и специального программного обеспечения промежуточного уровня (middleware), а также набора стандартизованных сервисов (служб) для обеспечения надежного совместного доступа к географически распределенным информационным и вычислительным ресурсам: отдельным компьютерам, кластерам, хранилищам информации и сетям. Основными направлениями развития грид-технологий являются: вычислительный грид, грид для интенсивной обработки данных и семантический грид для оперирования данными из различных баз данных.

Для построения полностью функциональной грид-системы необходимо программное обеспечение промежуточного уровня (middleware), построенное на базе существующих инструментальных средств и предоставляющее высокоуровневые сервисы задачам и пользователям. Создание и реализация грид-технологий является сложной научной и практической проблемой, находящейся на стыке большого количества научно-технических направлений.

**Типы грид-систем**

В настоящий время выделяют три основных типа GRID-систем:

* GRID на основе использования добровольно предоставляемого свободного ресурса персональных компьютеров (добровольная GRID);
* Научная GRID — хорошо распараллеливаемые приложения программируются специальным образом (например, с использованием Globus Toolkit);
* GRID на основе выделения вычислительных ресурсов по требованию (Enterprise GRID или коммерческая GRID) — обычные коммерческие приложения работают на виртуальном компьютере, который, в свою очередь, состоит из нескольких физических компьютеров, объединённых с помощью GRID-технологий.

**Пример использования**

Пусть есть ученый, которого попросили проверить чью-то новые полученные данные. Он начинает необходимые вычисления на удаленном сервере С. Начав исполняться, процесс на сервере С понимает, что ему необходима симуляция. Для этого он обращается к посреднику (на сервере D). Тот инициирует вычисления на серверах E и G, которые для получения необходимых параметров обращаются к серверу F. Данный пример иллюстрирует следующие важные характеристики грид-систем. Множество пользователей, как и набор ресурсов может быть большим и динамически меняться. Участники научного сообщества могут быть из разных организаций и меняться довольно часто, ресурсы могут изменять свои свойства. Вычисления могут охватывать разное число ресурсов в течение своего исполнения. Процессы исполнения могут общаться друг с другом, используя множество различных механизмов. Ресурсы могут потребовать разные механизмы аутентификации и авторизации и разных политик. Так в примере показаны следующие политики контроля доступа: Kerberos(сервер С), SSL(сервер D), обычные пароли. Также один и тот же пользователь может быть представлен на разных серверах разными именами, правами и учетными данным.