Высокопроизводительные параллельные вычисления

Лекция №1
Тема: Принципы построения параллельных вычислительных систем

Козбакова Айнур Холдасовна– PhD, профессор, кафедра «Информационные системы» Факультет «Инжиниринг и информационных технологии» 01.09.2025 г.

Под *параллельными вычислениями* понимаются процессы обработки данных, в которых одновременно могут выполняться несколько операций компьютерной системы. 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 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.

1.1 Классификация вычислительных систем

Рассмотрим классификацию вычислительных систем. Одним из наиболее распространенных способов классификации ЭВМ является систематика Флинна (Flynn), 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:

- SISD (Single Instruction, Single Data) – системы, в которых существует одиночный поток команд и одиночный поток данных. К такому типу можно отнести обычные последовательные ЭВМ;
- SIMD (Single Instruction, Multiple Data) – системы с одиночным потоком команд и множественным потоком данных. Подобный класс составляют *многопроцессорные вычислительные системы*, в которых в каждый момент времени может выполняться одна и та же команда для обработки нескольких информационных элементов; такой архитектурой обладают, например, многопроцессорные системы с единым устройством управления. Этот подход широко использовался в предшествующие годы (системы ILLIAC IV или СМ–1 компании Thinking Machines), в последнее время его применение ограничено, в основном, созданием специализированных систем;
- MISD (Multiple Instruction, Single Data) – системы, в которых существует множественный поток команд и одиночный поток данных. Относительно этого типа систем нет единого мнения: ряд специалистов считает, что примеров конкретных ЭВМ, соответствующих данному типу вычислительных систем, не существует и введение подобного класса предпринимается для полноты классификации; другие же относят к данному типу, например, *систолические вычислительные системы* (см. [1, 2]) или системы с конвейерной обработкой данных;
- MIMD (Multiple Instruction, Multiple Data) – системы с множественным потоком команд и множественным потоком данных. К подобному классу относится большинство параллельных *многопроцессорных вычислительных систем*.

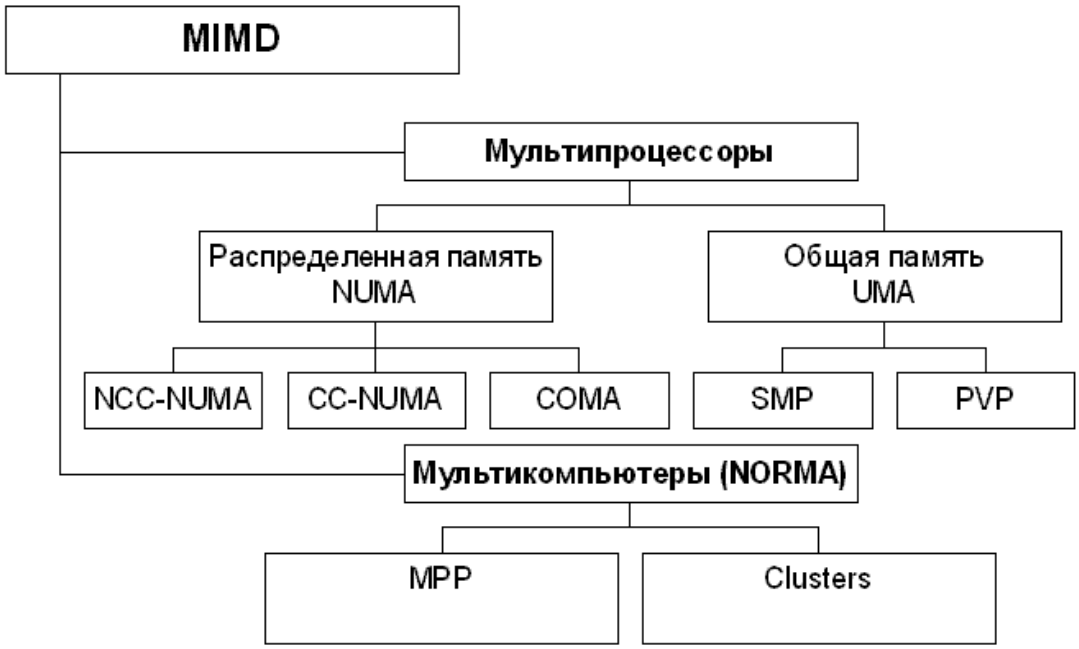

Рисунок 1.1 – Классификация многопроцессорных вычислительных систем

Следует отметить, что хотя систематика Флинна широко используется при конкретизации типов компьютерных систем, такая классификация приводит к тому, что практически все виды параллельных систем (несмотря на их существенную разнородность) оказываются отнесены к одной группе MIMD. Как результат, многими исследователями предпринимались неоднократные попытки детализации систематики Флинна. Так, например, для класса MIMD предложена практически общепризнанная структурная схема, в которой дальнейшее разделение типов многопроцессорных систем основывается на используемых способах организации оперативной памяти в этих системах. Такой подход позволяет различать два важных типа многопроцессорных систем – *multiprocessors* (*мультипроцессоры* или системы с общей разделяемой памятью) и *multicomputers* (*мультикомпьютеры* или системы с распределенной памятью).

1.2 Мультипроцессоры

Для дальнейшей систематики *мультипроцессоров* учитывается способ построения общей памяти. Первый возможный вариант – использование единой (централизованной) *общей памяти* (*shared memory*) (рисунок 1.2а). 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.

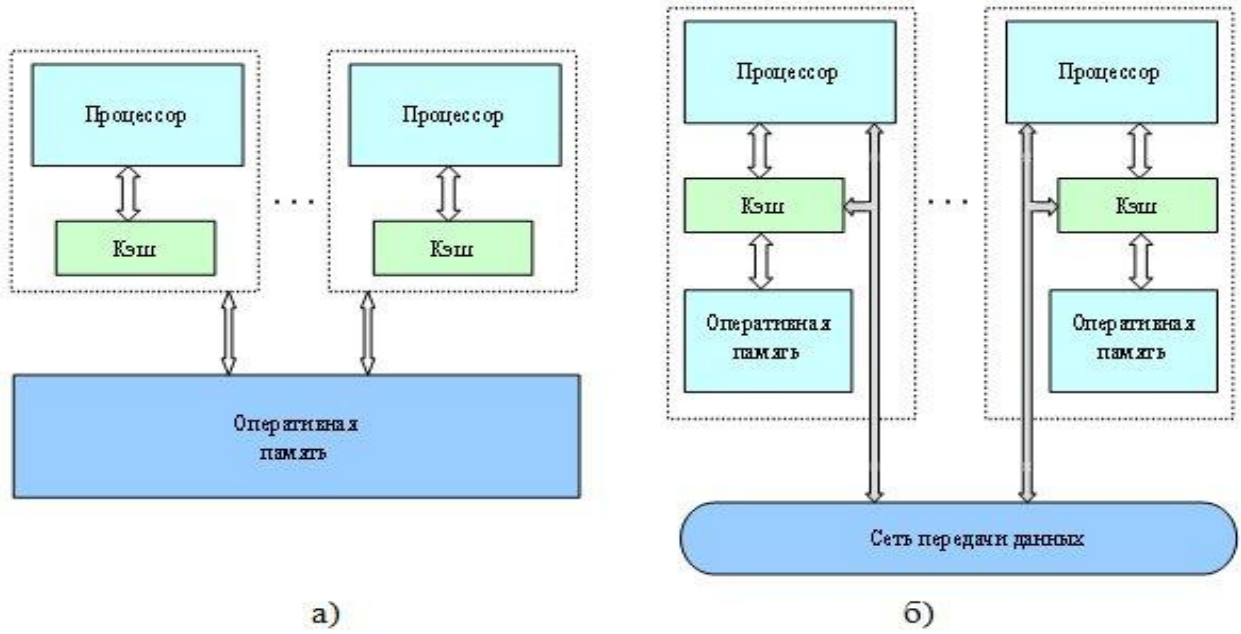

Рисунок 1.2 – Архитектура многопроцессорных систем с общей (разделяемой) памятью: системы с однородным (а) и неоднородным (б) доступом к памяти

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) (рисунок 1.2 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.

Использование распределенной общей памяти (*distributed shared memory* или *DSM*) упрощает проблемы создания *мультипроцессоров* (известны примеры систем с несколькими тысячами процессоров), однако возникающие при этом проблемы эффективного использования распределенной памяти (время доступа к локальной и удаленной памяти может различаться на несколько порядков) приводят к существенному повышению сложности параллельного программирования.

1.3 Мультикомпьютеры

Мультикомпьютеры (многопроцессорные системы с распределенной памятью) уже не обеспечивают общего доступа ко всей имеющейся в системах памяти (*no–remote memory access* или NORMA). При всей схожести подобной архитектуры с системами с распределенной общей памятью *мультикомпьютеры* имеют принципиальное отличие: каждый процессор системы может использовать только свою локальную память, в то время как для доступа к данным, располагаемым на других процессорах, 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)*. Среди представителей первого типа систем – IBM RS/6000 SP2, Intel

PARAGON, ASCI Red, транспьютерные системы Parsytec и др.; примерами *кластеров* являются, например, системы AC3 Velocity и NCSA NT Supercluster.

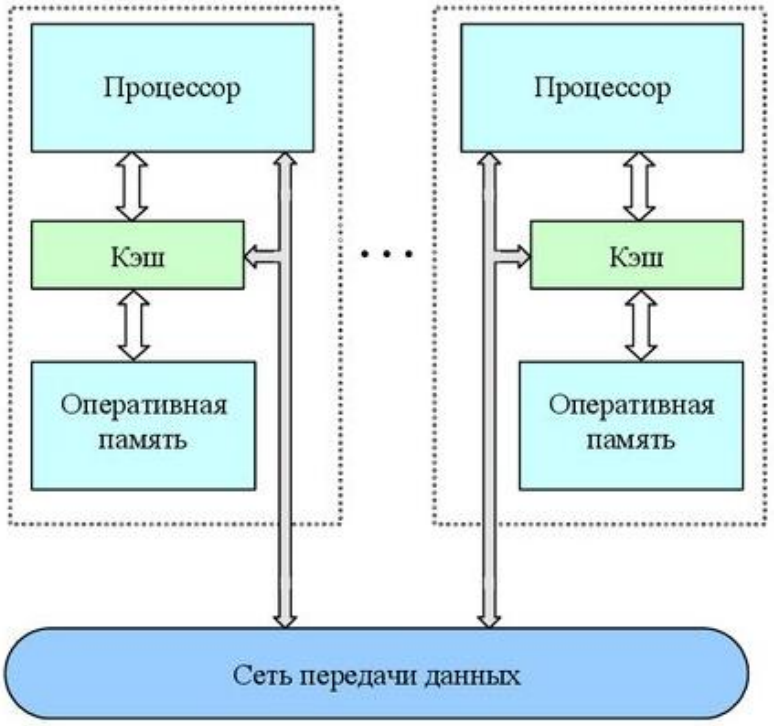

Рисунок 1.3 – Архитектура многопроцессорных систем с распределенной памятью

Следует отметить чрезвычайно быстрое развитие *многопроцессорных вычислительных систем* кластерного типа. 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, что обычно не требует значительных финансовых затрат.

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.

Отдельные исследователи обращают особое внимание на отличие понятия *кластера* от *сети компьютеров* (*network of workstations* или *NOW*). Для построения локальной компьютерной сети, как правило, используют более простые *сети передачи данных* (порядка 100 Мбит/сек). Компьютеры сети обычно более рассредоточены, и пользователи

могут применять их для выполнения каких–либо дополнительных работ. В завершении обсуждаемой темы можно отметить, что существуют и другие способы классификации вычислительных систем (достаточно полный обзор подходов представлен в [3]

Контрольные вопросы:

1. Что такое параллельная вычислительная система?
2. В чем отличие параллельных и последовательных вычислений?
3. Какова цель параллельных вычислений?
4. Какие основные типы вычислительных систем выделяются по классификации Флинна?
5. Чем отличается SISD от SIMD архитектуры?
6. Что такое MIMD-система?
7. Какие существуют подтипы MIMD-систем?
8. В чем разница между централизованной и распределённой памятью?
9. Что такое конвейерная обработка данных?
10. Какие критерии используются для классификации вычислительных систем?
11. Что такое мультипроцессорная система?
12. В чем разница между симметричными и асимметричными мультипроцессорами?
13. Что такое общая память в контексте мультипроцессорных систем?
14. Какие проблемы возникают при совместном доступе к общей памяти?
15. Что такое кэш-когерентность?
16. Какие основные методы обеспечения когерентности кэша существуют?
17. Какие способы синхронизации процессов в мультипроцессорах существуют?
18. Что такое NUMA-архитектура?
19. Как влияет масштабирование на производительность мультипроцессоров?
20. В чем преимущества и недостатки мультипроцессорных систем?
21. Что такое мультикомпьютерная система?
22. Как мультикомпьютеры взаимодействуют друг с другом?
23. Что такое распределённая память в мультикомпьютерах?
24. Какие типы топологий используются в мультикомпьютерах?
25. Как осуществляется маршрутизация сообщений между узлами?
26. Что такое MPI и зачем он используется?
27. В чем разница между мультикомпьютером и кластерной системой?
28. Какие преимущества дает масштабируемость мультикомпьютеров?
29. Какие проблемы возникают при разработке программного обеспечения для мультикомпьютеров?
30. Какие существуют примеры современных мультикомпьютерных архитектур?