Тема №2 " Структуры параллельных и распределенных КС. Параллельные алгоритмы. Представление, построение и анализ. Методы оценки производительности параллельных алгоритмов и систем "

вопрос:

1. Основные положения параллельных и распределенных компьютерных систем

2. шинные сети

3. Сети с коммутаторами

4. Структуры, обеспечивающие связь типа «точка-точка»

5. методы коммутаций

6. Общие замечания относительно оценки производительности параллельных систем

7. Факторы, необходимо учитывать при оценке производительности

8. Методы оценки производительности параллельных систем

9. Характеристики производительности параллельных алгоритмов

10. Сравнение MIMD і SIMD структур по производительности Упражнения и задания к лекции №2

1. Основные положения параллельных и распределенных компьютерных систем

Каждая параллельная система / процессор состоит из ряда процессоров и модулей памяти, которые связаны через соответствующие коммутационные структуры. Всі концепции коммуникаций, такие как "schared memory" (общая память, реальная или виртуальная), или обмен сообщениями реализуются в параллельных системах некоторыми имеющимися структурами. Учитывая задачи вычислительной системы, ее структура связи должна соответствовать *различным критериям,* прежде всего, *обеспечивать* по возможности *высокую коннективность (* гарантировать связь между двумя любыми процессорами или модулями памяти без необходимости перехода через промежуточные пункты коммутации). Окрім этого, должно обеспечиваться *наибольшее количество одновременных связей, -* чтобы коммутационная сеть не ограничивала производительность узлов параллельной обработки информации. Однако объективно существуют различные ограничения на реализацию этих критериев: *количество линий связи на один процессор* не может увеличиваться бесконечно, имеет также физические ограничения і *ширина частотной полосы пропускания (* скорость передачи) коммутационной сети.

Для параллельной системы, которая состоит из процессорных элементов (ПЭ) можно определить такие виды расходов:

- *количество связей на один ПЭ (затраты на изготовление системы);*
- *дистанция между ПЭ (расходы во время эксплуатации).*

То есть, количество связей на один процессор і дистанция, то есть кратчайший путь между двумя заданными ПЭ, должны быть как можно меньше. Структура связи должно быть расширяемой по масштабу, то есть малые сети связи должны увеличиваться благодаря дополнением.

Структуры связи делятся на три больших класса:

- шинные сетки
- сетки с коммутаторами;
- структуры, обеспечивающие связь типа "точка-точка".

Причем, для всех этих структур основной характеристикой сети остается *пропускная способность,* единицей измерения которой является *бит / с.*

Приведем основные *определение,* относящиеся к структуре связей.

топология - организация внутренних коммуникаций вычислительной системы. Два узла является *соседними,* если между ними есть прямые соединения.

порядком узла называется количество его соседей (обозначается V - количество связей).

коммуникационным диаметром сети является минимальный путь между двумя соседними узлами (обозначается A - расстояние).

масштабируемость характеризует рост сложности соединений при добавлении в конфигурацию новых узлов. (При произвольной масштабируемости системы, ее сложность при наращивании будет незначительно меняться, неизменным будет диаметр сети)

Есть два типа технологии коммутации сетей - *статические* и *динамические*.

2. Шинные сети

В параллельных структурах средствами шины можно соединить между собой процессоры или модули памяти (рис.5.1).

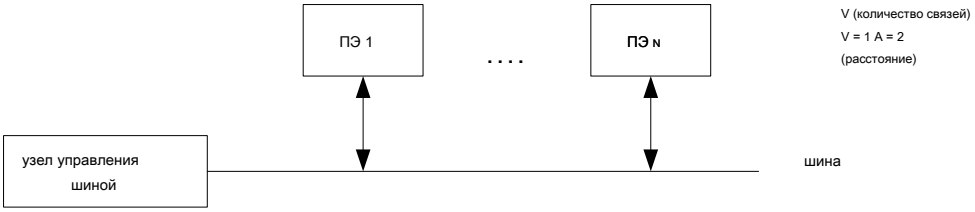


Рис. 2.1. шинная система

Количество связей на один ПЭ является оптимальной, расстояние между ПЭ является неизменной і равно 2 (каждый связь идет от ПЭ к шине і назад). Это также практически оптимально число. *Недостатки шинной организации* обусловлены тем, что в каждый момент может происходить только одно соединение. Параллельное чтение по одной и той адресу узла памяти возможно, однако запись - нет. Процессоры не могут независимо обмениваться данными; параллельная обработка при обмене данными между процессорами невозможно.

Система управления шиной должно гарантировать последовательное упорядочение одновременных запросов на услуги шины.

Если система может расширяться на многие процессоров, то ширина частотной полосы пропускания шины остается неизменной. Этот важный недостаток определяет масштабные способности шинных структур. Поэтому шинные системы как средства связи в параллельных ЭВМ, в которых количество процессоров превышает десять, применять не рекомендуется.

3. Сети с коммутаторами

Сети с коммутаторами - это динамические структуры связи, в которых с помощью линий управления реализованы различные варианты общения процессоров перед началом выполнения параллельной программы. Рассмотрим динамические сетки типов: *распределителя перекрестных шин, дельта-сетки, сети связи Клоса и сетки типа Fat-Tree.* На рис.2.2 приведена схема

распределителя перекрестных шин.

Каждый ПЭ имеет *n-1* пункт коммутации, поскольку диагональные пункты не требуют контакта. В общем сетка имеет *n (n-1)* пунктов коммутации і дает возможность установить любое количество соединений между всеми ПЭ, то есть без всяких коллизий может происходить полностью параллельный обмен данными. Существенным недостатком этого коммутатора являются затраты на его реализацию, составляют (*n* 2 *n)* пунктов соединений для *n* ПЭ. Практически такое количество пунктов может быть реализована только для небольшого количества процессоров. Уже для 100 ПЭ нужно 9900 переключений, чтобы обеспечить работу всіх пунктов соединений.

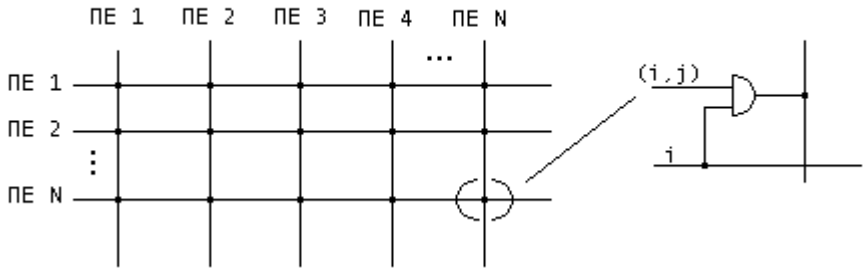


Рис. 2.2. Распределитель перекрестных шин

Дельта - сети

Чтобы уменьшить расходы, возникающие при построении распределителей перекрестных шин, используются дельта-сетки (см. Рис.2.3).

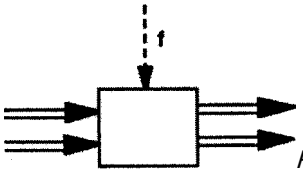


Рис. 2.3. Дельта-сети коммутации

В простейшем случае (рис.2.3) две линии данных могут переключаться накрест с помощью единой линии управления или на прямую передачу с входа на выход.

На рис.2.4 показано, как проходят сигналы внутри "черного ящика" некоторой дельтаситкы. Если управляющий сигнал равен "нулю", то обе линии данных (или пучки линий) присоединяются прямо к исходных линий. В случае, когда управляющий сигнал равен "единицы", линии данных переключаются на выходе накрест.

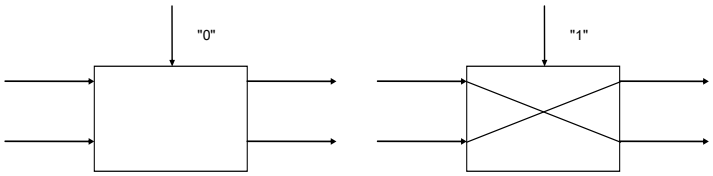


Рис. 2.4. Переключатель дельта-сети

3 таких простых базовых элементов можно строить более объемные сетки. На рис.2.5 приведены трехступенчатую дельта-сетку, которая соединяет 8 входов с 8 выходами. Каждый элемент сетки соответствует базисному элемента показан на рис.2.3.

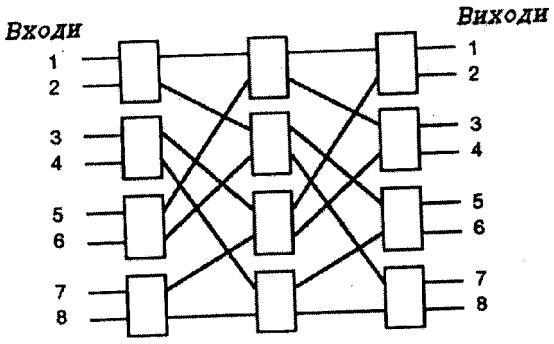


Рис. 2.2. Дельта-сетка размером 8x8

Преимуществом дельта-сетей над распределителями перекрестных шин является меньшее количество дельта

переключателей, — * log *пп.* **главным *недостатком* является то, что не всі возможные комбинации связей между**
2

процессорами могут быть реализованы. Здесь могут возникать блокировки для предотвращения которым нужно использовать специальные концепции программирования. Блокировка существенно задерживает выполнение параллельной программы, обусловлено необходимостью переконфігураціи коммутирующей сети и построения новой схемы соединений между процессорами. Эта ситуация аналогична той, что имеет место в телефонных сетях.

Коммутирующие сети Клоса

Объединение лучших свойств распределителя перекрестных шин и дельта-сетей сделано в коммутирующих сетях Клоса. Требованием к этой сетевой структуры является обеспечение произвольной комбинации связей между процессорами, то есть не допускается появление блокировок. З другой стороны, расходы на реализацию (количество пунктов коммутации) должны быть минимальными. Это достигается построением минимизированной по затратам многоступенчатой сети. Причем элементы одной ступени строятся на основе простых малых распределителей перекрестных шин. На рис.2.6 показано трехступенчатую сеть Клоса из N = 12 входами, распределенными на a = 4 группы, каждая из которых имеет m = 3 входов.

В трехступенчатой сети Клоса общее количество *N* линий переключаются полностью реализуется в первой степени *a* малы
распределителями перекрестных шин, каждый из которых имеет

m входов і *2 * (m-1)* выходов (*N = a * m).* Вторая ступень построена на *2 * (m-1)* распределитель перекрестных шин, каждый из которых имеет *a* входов і *a* выходов, а третья - на распределителем с *2 * (m-1)*
входами і *m* выходами. Параметром, который может выбираться при конфигурировании сети Клоса, является, таким образом, *m.* Общее количество пунктов коммутации трехступенчатой сети Клоса будет примерно минимальной, если *m* определить по формуле:

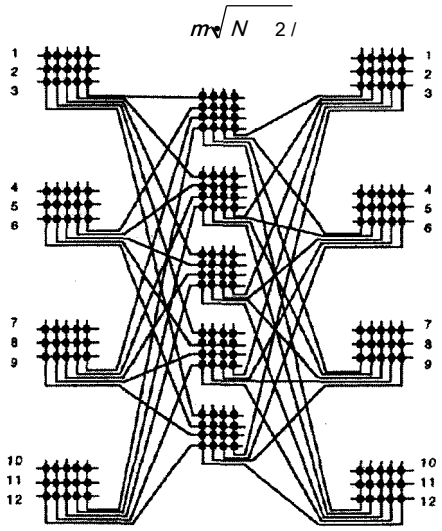


Рис. 2.6. Трикаскадна сеть Клоса для N = 12 (при a = 4, m = 3)

при N • 24 сеть Клоса эффективнее, чем простой распределитель перекрестных шин. Общее количество пунктов коммутации в трехступенчатой сети Клоса определяется по формуле:

Она содержит (2 * а) распределители перекрестных шин размером *"M: (2 * m-1)"* на ступенях 1, 3 і (*2 * m-1)* распределителя размером " *а именно: a "* на степени 2. 3 учетом того, что *N = a * m,* имеем:

Если количество входов *m* оптимальная, то расходы на трехступенчатую сеть Клоса составляют примерно

В противоположность этому расходы на полный распределитель перекрестных шин соответствующего размера составляют примерно *N2.*

Пример.

Для трехступенчатой сети Клоса с 1000 входами и выходами справедливо:

берем ближайшее *m = 20,* что дает *a = N / m =* 50. Общее количество пунктов коммутации будет:

Это количество значительно меньше, чем в полном распределители перекрестных шин этого размера:

Таким образом, сеть Клоса экономит более 82% пунктов коммутации по сравнению с распределителем перекрестных шин. Размеры распределителей перекрестных шин, применяемых в качестве элементов сети Клоса, выбирают в соответствии с количеством коммутируемых процессоров і это исключает возможность блокировки связей.

Сети типа Fat-Tree (родительское дерево)

Fat-Tree - решетчатая структура, которая может масштабироваться как по количеству процессоров, так і по количеству возможных соединений, происходящих одновременно. Процессоры - это листья полного двоичного дерева, в то время как внутренние узлы - это ключевые элементы (рис.2.7).

Количество процессоров - $N = 2^m$, а количество ключей переключения равна $N-1$. Чем выше содержится ключевой элемент в дереве, тем больше линий связи переключается в нем (конечно, нет потребности в удвоенном количестве линий на каждом рівни дерева).

Коммутационная структура Fat-Tree близка к оптимальной. Это свойство делает ее очень интересной для применения в параллельных структурах.

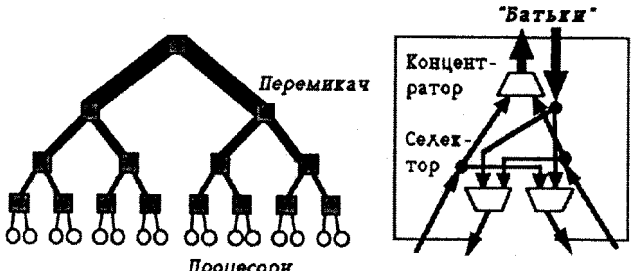


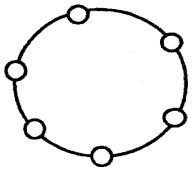
Рис. 2.7. Сеть типа Fat-Tree и переключатель

4. Структуры, обеспечивающие связь типа "точка-точка"

Рассмотрим статические структуры связи "точка-точка".

кольцо

Кольцевая структура (рис.2.8) имеет две линии связи на каждый ПЭ (***преимущество)*** **і требует в худшем случае $n / 2$ шаги** для обмена данными между двумя ПЭ, размещаются в кольце на наибольшем расстоянии друг от друга (*недостаток).*

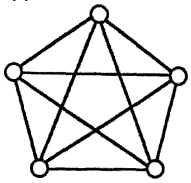


V = 2 A = n / 2

Рис. 2.8. кольцо

полный граф

Полный граф (рис.2.9) имеет оптимальную коннективность (каждый ПЭ связан непосредственно с любым **другим ПЭ) обеспечивается $n-1$ линией связи на каждый ПЭ.**

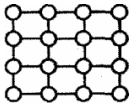


V = n-1 A = 1

Рис. 2.9. полный граф

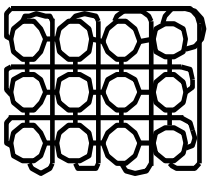
Решетки і торы

Очень часто применяются решетчатые структуры связи і их замкнутые варианты - торы. На рис.2.10 показано различие между четыре - і восьми - связными структурами.

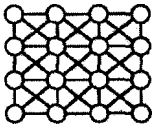


квадратная решетка (4 линии связи в ПЭ) V = 4; A = $2 * n - 2 = 2 * \sqrt{n}$

Кваратний тор (4 линии связи ПЭ)

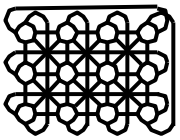


V = 4; A = n



V = 8; A = $n - 1 = n$

Кваратний тор (8 линии связи ПЭ)



V = 8 A = $2\sqrt{n}$

квадратная решетка

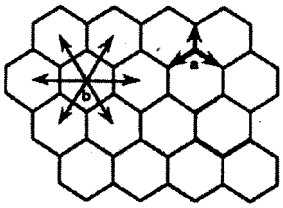
(8 линий связи в ПЭ)

Рис. 2.10. Квадратные решетки и торы

Все квадратные решетки имеют максимальное расстояние между ПЭ, равна квадратному корню от количества ПЭ. Увеличение количества линий вдвое в восьмизвьязковий решетке делает расстояние между ПЭ наполовину меньше. Такой же эффект достигается переходом к замкнутому тора, в котором количество ПЭ остается без изменения.

Гексагональная решетка.

Гексагональная решетка может рассматриваться как видоизменение квадратной решетки. В зависимости от того, где размещены процессорные элементы (на перекрестке или в центре, рис.2.11 а, б), они нуждаются в 3 или 6 линий связи.



а) ПЭ в углу

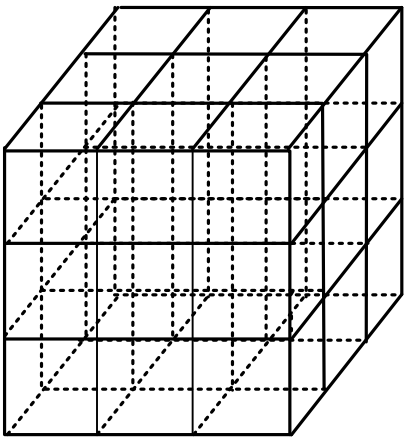
*V = 3 A = 2 * n*

б) ПЭ в центре

*V = 6 A = 3/2 * n Рис. 2.11. гексагональная решетка*

кубическая решетка

В кубической решетке (рис.2.12) сделано переход от двумерной к трехмерной решетки. Процессорные элементы размещены в пространстве куба требуют 6 линий связи из ПЭ - соседями. Максимальное расстояние между ПЭ здесь уменьшается, она пропорциональна $\sqrt[3]{n}$.

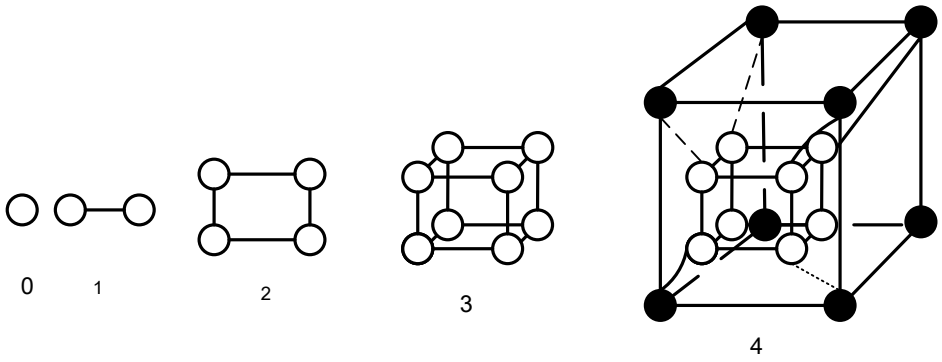


$V = 6; A = 3 * \sqrt[3]{n}$

Рис. 2.12. кубическая решетка

гиперкуб

Гиперкуб нулевой размерности - это единственный элемент (рис.2.13, слева). гиперкуб размерности $i + и$ возникает из двух гиперкубов размерности *и,* в которых элементы, взаимодействуют, соединенные между собой. Например, на рис.2.13 показано, что из двух квадратов (гиперкубов размерности 2) можно построить куб (гиперкуб размерности 3), соединив каждый элемент "переднего" с каждым элементом "заднего" квадрата. Гиперкуб является универсальной сетью связи с небольшим логарифмической расстоянием і не слишком большой логарифмической количеством линий связи у каждого ПЭ.



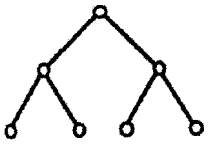
$n = 2^m; V = \log_2 n$

$A = \log_2 n$

Рис.2.13. Гиперкуб с размерностями от нуля до четырех

двоичное дерево

Следующая структура связи, имеет логарифмическую расстояние между ПЭ, это двоичное дерево (рис.2.14). Недостатком структуры является "узкое место корня", которое ограничивает обмен данными между парами ПЭ из разных поддеревьев подобно тому, как это делает шинная система. Некоторой мірою этот недостаток уменьшает применение Fat-Tree.



$N = 2^m - 1\ V = 3\ A = 2 * \log_2 (n + 1) - 2 \cdot 2 * \log_2 n$

Рис. 2.14. двоичное дерево

пирамидальное дерево

В этой структуре связи (рис.2.15) каждая вершина имеет 4 последователи в дереве что, может использоваться для построения алгоритмов распределения образов в квадрантах.

$N = (1/3) * (4^m - 1)\ V = 5\ A = 2 * \log_4 (3 * n + 1) - 2 \cdot 2 * \log_4 n$

Рис. 2.15. *пирамидальное дерево*

Ротация-изменение. (Shuffle-Exchange)

К структурам с логарифмической размерностью принадлежит также сеть Shuffle-Exchange (ротация, или циклический сдвиг и замена). Она состоит из двух разделенных видов связи одностороннего "Shuffle" и двустороннего "Exchange" (рис.2.16).

$N = 2^m$

$V = 2$ (1 двунаправленная и 2 однонаправленные линии)

$A = 2\log_2 n$

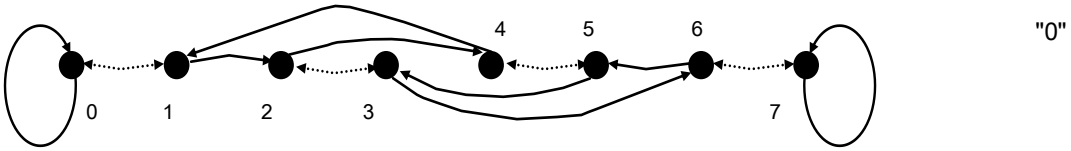


Рис.5.16. Сеть типа "Shuffle-Exchange"

Обе структуры связи могут быть очень просто представлены с помощью операции отображения номеров ПЭ **бинарным способом записи (** p_i **- бит бинарного номера ПЭ,** $i = i\ 2 .. m$**):**

Shuffle ($p_m, p_{m-1}, ..., p_1$) = ($p_{m-1}, , p_1, p_m$) - одна ротация влево бинарного номера ПЭ (*m* раз)

Exchange ($p_m,$ p_1) = ($p_m, p_2 p_1$) - замена младшего бита бинарного номера ПЭ операцией возражения.

Shuffle - часть соединяет каждый ПЭ с тем ПЭ, бинарный номер которого исчисляется циклической операцией ротации влево. Первый и последний элементы, имеющие одинаковые двоичные разряды в номере (на рис.2.16 это ПЭ *No • 000, ПЭ N7 • 111)* отражаются сами на себя, тогда как другие ПЭ соединены в циклы.

Exchange - связь в двоичной формі записи отрицает молодой бит ПЭ, то есть Exchange соединяет двунаправлено каждый парный ПЭ со своим правым соседом.

Пример применения операций Shuffle и Exchange:

1) *001 • 010 • 100 • 001*

2) *011 • 010 • 011* двустороннюю связь ПЭ3 • ПЭ2

Плюс-минус 2i- сеть (PM2и)

Сеть несколько сложнее. при наличии $n = 2^m$ узлов (вершин) сети (процессорных элементов ПЭ) она состоит из $2 * r-1$ отдельных структур связи, которые обозначаются как:

$$PM+_0, PM_0, PM+_1, PM-_1, PM+_2, PM-_2, ..., PM_{m-1},.$$

для *PMs* справедливы следующие определения:

$$PM+_1(j) = (j + 2^i) \bmod n; \quad PM-_1$$

$$(j) = (j-2^i) \bmod n.$$

Индекс каждой односторонней структуры показывает, какой должна быть расстояние до соседней вершины (в двоичном порядке). Для PM +0 расстояние будет + 2 ° = + 1, для Pm0 соответственно -2 ° = 1 Расстояние между вершинами для PM +2 при этом есть $2^2 = +4$, для PM.-2 будет - 4. Для высшего показателя $r-1$ с учетом замкнутости структуры имеет место соотношение: $PM+(m-1) = PM-$

(M-1)

Обе структуры идентичны, поэтому есть только $2 * r-1$, а не $2 * r$ структур. На рис.2.17 показано сеть *PM2I* с n = 8 ПЭ. PM +0 i Pm0 связывают каждый ПЭ со своим правым или левым соседом і создают таким образом двустороннее кольцо связи. , PM +1, и PM.-1,

включают две разделенные односторонние структуры по четыре ПЭ. Каждый ПЭ имеет соседа с номером через один, причем *Modulo* - операция образует структуру связи типа "кольцо". Вместе они образуют два отдельных двусторонних кольца. Последняя структура CM +2 идентична с PM.-2,.

Каждый ПЭ связан здесь с другим ПЭ на расстоянии 4 (*Modulo 8).*

$V = 2 * \log_2 n-1$ $A = \log_2 n-1$

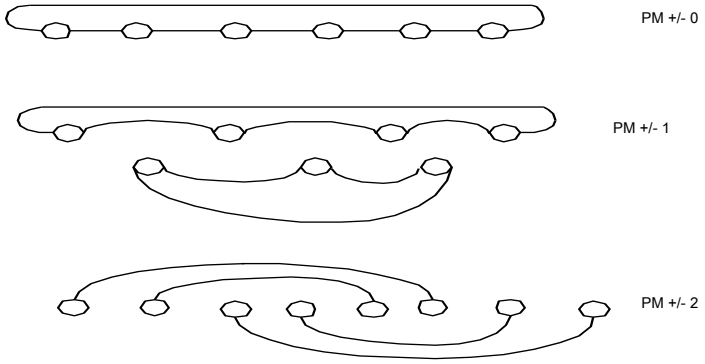


Рис. 2.17. Сеть типа PM2I

сравнение сетей

Сравнение сетей возможно только на основе параметров *V* i *A.* Преимущества, имеет определенное сеть, можно получить только с учетом особенностей параллельной задачи. Если алгоритм решения задач требует определенной сети, то, как правило, физическая реализация как раз этой быть "хуже" по параметрам *V* i *A* сети, даст лучшие результаты, чем приспособления для этой задачи другой сети, имеющей "хорошие" параметры.

Для решения каждой проблемы невозможно строить специализированную систему. Каждая параллельная система имеет в своем распоряжении несколько коммутационных структур, которые могут быть многосторонними і способными к динамической конфигурации. Все алгоритмы, которые подлежат имплементации в этой параллельной системе, должны удовлетворяться имеющейся сетью (или ее частью). Ценность сети зависит, таким образом, от того, насколько качественно она может использоваться в среднем для решения проблем, возникающих чаще всего. Например, параллельная ЭВМ MasPar MP-1 функционирует с применением двух различных сетей: решетчатой структуры и трехступенчатой сети Клоса, что обеспечивает хорошую общую коннективность. Алгоритмы, которые нуждаются в решетчатой структуры, могут использовать непосредственно эту быструю локальную структуру, тогда как алгоритмы,

Зигель моделировал одни сети с применением других сетей. При этом оказывается прежде всего преимущество Shuffle-Exchange и гиперкуба *как "*универсальных сетей ", то есть как средств эффективной эмуляции нескольких різноманитних сетевых структур. Решетка только тогда оправдана, когда решетчатая структура нужна для реализации алгоритмов, для моделирования других сетей она непригодна. Прежде всего эмуляция сети может быть выполнена автоматически компилятором только тогда, когда кроме сетевой структуры целевой системы известна также структура, которая нужна для реализации приложения.

Чаще всего реляции о межпроцессорный связи, которые должны обеспечить обмен данными, заданные в виде сложных арифметических выражений, мало отличаются от "стандарта" или вообще исчисляются і становятся известными только перед запуском программ. В табл.2.1 показано количество нужных циклов моделирования в зависимости от количества ее процессорных элементов.

Таблица 2.1 . сравнение к омутацийних сетей

сеть Моделирует сеть	двумерная решетка	PM2 I	Shuffle-Exchange	гиперкуб
двумерная решетка	___	• $\sqrt{n}$ / 2	• $\sqrt{n}$	$\sqrt{n}$
PM2I	1	___	• log $_2$ n	2
Shuffle-Exchange	• log 2_2 n	• log 2_2 n	___	• log $_2$ n + 1
гиперкуб	log $_2$ n	log $_2$ n	log $_2$ n	___

5. Методы коммутаций

Важным в организации работы коммутационной сети является метод переключения, который определяет как сообщения передаются от узла-передатчика до узла-приемника. передача осуществляется

сообщениями и *пакетами.*

Выделяют три основных метода коммутации в коммутационных сетях параллельных систем:

- коммутация с промежуточным хранением ("хранения- и передача" - Store-and-forward)
- коммутация каналов;
- коммутация виртуальных каналов.

В первом случае сообщение полностью принимается на промежуточном узле и только после этого передается дальше. Пересылка выполняется как только освободился очередной канал передачи, тогда сообщение передается на следующий узел. Буферизация требует дополнительной памяти и затрат времени. Метод используется тогда, когда время отклика не существенно.

При коммутации каналов между источником и получателем сообщения устанавливается непрерывную связь, после чего выполняется передача данных. Коммутирующий канал устанавливается только на время соединения. Пример - телефонная связь.

Уменьшить задержки при передаче данных позволяет метод виртуальных каналов. В этом случае пакеты накапливаются в промежуточных узлах только тогда, когда недоступен очередной канал связи. В противном случае пересылка выполняется немедленно и без буферизации.

6. Общие замечания относительно оценки производительности параллельных систем

Параллельные системы характеризуются: *разноплановистю задач, типу обработки (векторное, скалярное), различными операционными системами, различными конфигурациями систем, одновременностью выполнения операций, сложности взаимодействия между элементами системы,* что не позволяет использовать стандартные подходы к определению их производительности.

А производительность отдельного процессора зависит от: *частоты синхронизации, среднего количества тактов на команду, количества команд выполняется.*

Поэтому *время выполнения некоторой программы в процессоре может быть выражен двумя способами: количеством тактов синхронизации для данной программы, перемножаются на продолжительность такта синхронизации, или количеством тактов синхронизации данной программы разделенными на частоту синхронизации.*

В процессе поиска стандартной единицы измерения производительности компьютеров было принято несколько популярных единиц измерения, а для оценки производительности параллельных узлов некоторые термины были искусственно вырваны из их контекста и использованы там, для чего они никогда не предназначались. На самом деле *единственной и надежной единицей измерения производительности является время выполнения реальных программ,* и все предлагаемые замены этого времени как единицы измерения или замены реальных программ как объектов измерения на синтетические программы только вводят в заблуждение.

Опасности использования популярных альтернативных единиц измерения (MIPS и MFLOPS) для оценки производительности параллельных систем.

MIPS - *скорость выполнения операций в единицу времени,* то есть - отношение количества команд в программе ко времени ее выполнения. *MIPS:*

- *зависит от набора команд процессора, затрудняет сравнение по показателям MIPS компьютеров, имеющих различные системы команд*
- *даже на том же компьютере меняется от программы к программе;*
- *изменяться по отношению к производительности в противоположную сторону (при большем значении MIPS реальная производительность меньше).*

Пример для последнего случая.

В состав процессора входят *узлы вычисления элементарных функций.* при отсутствии *узлов* операции вычисления элементарных функций выполняются с помощью подпрограмм. Тогда, такие процессоры имеют более высокий рейтинг MIPS, но выполняют большее количество команд, приводит к увеличению времени выполнения программы.

MFLOPS - миллион операций с плавающей точкой в секунду. *В качестве единицы измерения, MFLOPS, предназначенная для оценки производительности только операций с плавающей точкой, и поэтому не применяется вне этой ограниченной области.*

К примеру, программы компиляторов имеют рейтинг MFLOPS близок к нулю независимо от того, насколько быстрая машина, поскольку компиляторы редко используют арифметику с плавающей точкой.

Рейтинг MFLOPS зависит и от характеристик процессора и от программы, и является объективным чем параметр MIPS, поскольку *базируется на количестве выполненных операций, а не на количестве выполненных команд.*

Однако и с использованием MFLOPS для оценки производительности возникают определенные проблемы. Прежде всего, это связано с тем, что *наборы операций с плавающей точкой не совместимы на разных компьютерах.* Например, в суперкомпьютерах фирмы Cray Research отсутствует команда деления (есть, правда, операция вычисления обратной величины числа с плавающей точкой, а операция деления может быть реализована посредством умножения делимого на обратную величину делителя). В то же время современные микропроцессоры имеют команды деления, вычисления квадратного корня, синуса и косинуса, и тому подобное.

Другая проблема заключается в том, что *рейтинг MFLOPS меняется не только на смеси целочисленных операций и операций с плавающей точкой, но и на смеси быстрых и медленных операций с плавающей точкой.* Например, программа по 100% операций сложения будет высокий рейтинг, чем программа со 100% операций деления.

Для устранения этих недостатков *используется "нормализованное" число операций с плавающей точкой,* по тестовым пакетом "Ливерморской циклы" (см. табл.2.2.).

Таблица 2.2. Соотношение между реальными и нормализованными операциями с плавающей точкой

реальные операции с подвижной точкой	Нормализованы операции с плавающей точкой
" • "" • "" X "" сравнения "	1
"деления" " • "	4
"Exp" "sin"	8

Появление векторных и параллельных процессоров и систем, не уменьшил важности Ливерморской циклов, однако изменились значения производительности и величины разброса между различными циклами. На векторных структурах производительность зависит не только от элементной базы, но и от характера самого алгоритма, то есть коэффициента векторизации.

На параллельной машине производительность существенно зависит от соответствия между структурой аппаратных средств и структурой вычислений в алгоритме. Важно, чтобы тестовый пакет представлял алгоритмы различных структур. Поэтому в Ливерморской циклах встречаются последовательные, сетевые, конвейерные, волновые вычислительные алгоритмы, подтверждающий их пригодность и для параллельных машин.

7. Факторы, необходимо учитывать при оценке производительности

При оценке производительности необходимо учитывать: *тип алгоритма, тип программного обеспечения, параметры протоколов каналов передачи данных, структуру отдельного процессора.*

Тип алгоритма. Алгоритм, идеально приспособлен для работы на одной архитектуре, на другой (с этой же количеством процессоров) может работать гораздо хуже. Для масивнопаралельних систем необходим масштабируемый алгоритм (для «оптимального» загрузка всех процессоров). Выигрыш дает оптимальное сочетание "структура - алгоритм". Есть, *на параллельной машине производительность зависит от соответствия между структурой аппаратных элементов и структурой вычислений в алгоритме. Нельзя переносить результат с одной на другую систему.*

Тип программного обеспечения. Программное обеспечение (ПО) параллельных структур имеет определенные особенности, а именно: *стоимость программ высока, при переносе программы с одной машины на другую необходима доработка программы, все системы отладки программы влияют на его поведение (например, пошаговое отладки для параллельных систем неэффективное).*

Кроме того, *программисту трудно научиться мыслить параллельными категориями.*

В состав ПО необходимо включать *процедуры маршрутизации.* для оценки производительности распределенной системы необходимо знать: *топологию связей, скорость выполнения арифметических операций, время инициализации канала связи, передаче единицы информации.*

Кроме того, нужно помнить, *что рост производительности процессоров опережает рост скорости коммутационных каналов.*

Протокол каналов передачи. Для параллельных машин целесообразно определить библиотеку передачи сообщений, которая учитывает особенности машины. В 1994 году принят стандартный интерфейс передачи сообщений *MPI (Message Passing Interface Standart)* - процедурный интерфейс для языков C и Fortran). Интерфейс определяет все функции, необходимые для передачи сообщений "точка-точка". Для коллективных сообщений вводятся понятия *группы процессоров*

(С которыми можно оперировать как с конечными множествами), *коммуникатора (* реализуют контекст для передачи сообщений). Обеспечивает трансляцию сообщений с формы одного процессора в вид, который необходим другом процессору. Не решена проблема динамической балансировки.

8. Методы оценки производительности параллельных систем

Есть такие методы оценки производительности параллельных систем: *метод вычисления производительности составных частей, метод экспертных оценок, расчетный метод, практический метод.*

Метод вычисления производительности составных частей. Для параллельных систем неэффективен.

Метод экспертных оценок. Самый сложный и найзаперечливиший из всех методов. Разработан консорциумом стандартизации методов всесторонних оценок системы (PC Bench Concortium). Особое внимание уделено *аппаратном подхода (* стараются подражать идеализированную тестовую модель - исключить влияние сторонних систем, которые в данный момент не тестируются).

используется *две группы тестов:*

- *синтетические (* измеряют скорость работы и базируются на хронометраже работы реальных приложений; легко изолируют процессор от остальных компонентов системы. К реальной задачи можно отнести с натяжкой)

- *исключительно процессорные тесты (* для процессоров отдельно).

Расчетный метод. Базируется на вычислении производительности. Есть трудоемким и несовершенным.

Практический метод. Решение конкретной задачи на конкретной структуре. Дает объективную оценку производительности. *недостаток* - производительность можно оценить только после изготовления системы. При отрицательном результате - высокая цена ошибки.

9. Характеристики производительности параллельных алгоритмов.

Характеристиками производительности параллельных алгоритмов являются: *фактор ускорения, максимальное ускорение (закон Амдала), эффективность параллельного алгоритма, цена, масштабность, общее время выполнения параллельного алгоритма, полное время выполнения параллельного алгоритма, теоретический время коммуникаций.*

Оценивая производительность непосредственно на параллельных системах, отмечают положительный эффект от *распараллеливания (Speedup*- ускорение) и выгоды от увеличения
масштабности решенных задач (Scaleup).

Показатель Speedup определяет, во сколько раз быстрее может быть решена одна и та же задача на N процессорах сравнению с ее решением на одном процессоре.

Показатель Scaleup определяет, во сколько раз больше по размерам проблему можно решить за то же время N процессорами в сравнении с проблемой, решаемой одним процессором.

Амдал сформулирован "закон Амдала", где единственным параметром является разделение программы на последовательную и параллельную части; масштаб решаемой задачи остается постоянным. Рассмотрим несколько упрощенную формулу этого закона.

Фактор ускорения. ускорением (*speedup factor)* параллельного алгоритма в N - процессорной системе называется величина $S(N) = T_1 / T_N$, где

- T_1 - время выполнения алгоритма на одном процессоре или однопроцессорной системе;

- T_N - время выполнения алгоритма на многопроцессорной системе с N - процессорами.

Закон Амдала. обозначим:

- P_c - максимальная степень распараллеливания (максимальное количество процессоров, которые могут работать параллельно в любой момент времени в период выполнения программы). Здесь решающее значение имеет также тип применяемой модели параллельности (например, MIMD или SIMD)

- T_k - продолжительность выполнения программ с максимальным показателем распараллеливания k P_c • на системе с k процессорами;

- N - количество процессоров в параллельной системе;

- f - процент последовательных операций программы (не могут быть выполнены параллельно на N процессорах).

Продолжительность программы на параллельной системе с N процессорами оценивается формулой:

$$\bullet \quad * \quad {}_{1} \bullet (1 \bullet \quad) * \frac{1}{NT}, \qquad fTfT_N$$

откуда получим показатель ускорения (Speedup) в системе с N процессорами

$$\bullet \frac{1}{TTS_{NN}} \bullet \frac{N}{1 \bullet \quad * (1 \bullet \quad Nf)}.$$

поскольку, $0 < f < 1$, справедливо такое соотношение:

$$1 \bullet \qquad N \bullet NS,$$

то есть показатель ускорения не может быть больше чем количество процессоров N.

В качестве меры достигнутого ускорения, относительно максимального определяется эффективность системы из N процессорами

$$\bullet \qquad NN \big/ NSE$$

Область пределы эффективности: $\bullet \; EN_N \bullet 1/1$

На практике используют значение N E в процентах. при N E = 0,9. например, могла бы быть достигнута эффективность, равная 90% максимально возможной.

примеры применения закона Амдала

1. Система имеет N = 1000 процессоров

- Программа имеет максимальный показатель распараллеливания 1000
- 0,1% программы выполняется последовательно (например, операции ввода-вывода), т.е.

f = 0,001 Расчет показателя ускорения дают:

$$S_{1000} \bullet \frac{1000}{1000 + 1 \bullet 1} \bullet 500$$

Таким образом, несмотря на существенно малую долю последовательной часть программы, в этом случае достигается только половина максимально возможного ускорения, то есть $S_{max} \bullet \quad 1000$.

8

эффективность будет $E_{1000} \bullet \quad 50\%$.

2. Система имеет N = 1000 процессоров

- Программа имеет максимальный показатель распараллеливания 1000.
- 1% программы должно выполняться последовательно, то есть f = 0,01 Показатель ускорения:

$$S_{1000} \bullet \frac{1000}{1000 + 1 \bullet 1} \bullet 91$$

эффективность E_{1000} = 9,1%. то есть используется только 9,1% общей мощности процессоров и это при малой, на первый взгляд, последовательной части программы!

С увеличением последовательной части программы падает загрузки процессоров, а с ним и показатель ускорения по сравнению с последовательной вычислительной системой. Для каждой последовательной части программы может быть вычислен максимально возможный показатель ускорения независимо от количества применяемых процессоров:

$$\bullet S_{NN} \quad (\lim_{N \to \infty}) \bullet \frac{N}{1 \bullet \quad * \quad N \to \infty f(\;)} \bullet \frac{1}{f}$$

Это означает, что программа по скалярной частью, которая составляет 1%, никогда не сможет достичь показателя ускорения (Speedup), который был бы большим 100 - независимо от того, применяется 100,1000 или 1000000 процессоров.

На рис.2.18 приведены графики зависимости показателя ускорения от количества процессоров N для различных величин последовательной части программы f. Главная диагональ относится к линейных ускорений и может быть построена при f = 0.

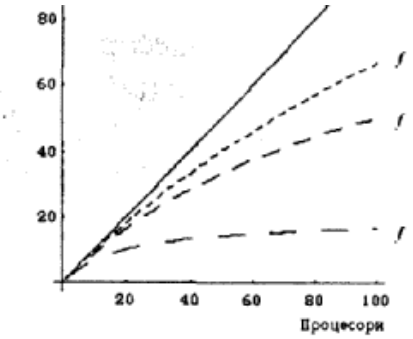


Рис. 2.18. Зависимость ускорения от количества процессоров

На рис. 2.19 приведены графики зависимости эффективности от *N* и f.

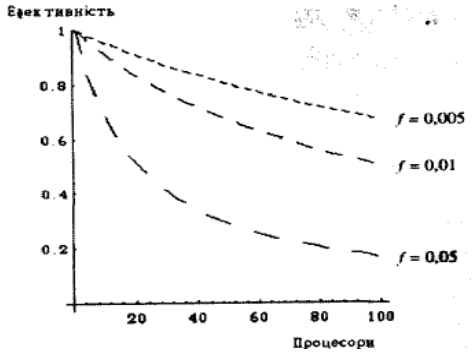


Рис. 2.19. Зависимость эффективности от количества процессоров.

эффективность параллельного алгоритма *E* показывает во сколько раз большее время выполнения задания одним процессором T_1, чем время выполнения этого же задачи многопроцессорной системой T_N, умноженный на количество процессоров *N*. $E = T_1 / (T_N \cdot N) = S_N / N$. Эффективность характеризует ту часть времени, которую процессоры используют на вычисления.

*Цена (*работа) вычисления определяется как *Cost = (время выполнения) * (полное число использованных процессоров).* Цена последовательных вычислений равна времени их выполнения T_1. Цена параллельных вычислений равна $T_N \cdot N$. С другой стороны $T_N = T_1 / S_N$. Отсюда цена параллельных вычислений: $Cost = T_1 \cdot N / S_N = T_1 / E$. Циноптимальни параллельные алгоритмы - алгоритмы, в которых цена для решения проблемы на многопроцессорной системе пропорционально цене (времени выполнения) на однопроцессорной системе.

Масштабность (Scaleup)

Поскольку для измерения эффективности применяется та же программа, то показатель *f* в законе Амдала остается неизменным. Однако, каждая параллельная программа, независимо от ее величины, имеет последовательно обрабатывать некоторую постоянную минимальное количество *f* своих инструкций. Это означает, что графики, которые приведены на рис.2.18 и рис. 2.19 с изменением размеров проблемы становятся недействительными! Практические замеры на параллельных системах с очень большим количеством процессоров показали: чем больше есть решаема проблема и количество используемых процессоров, тем меньше становится процент последовательной части вычислений. Это дает основание сделать вывод, что на параллельной системе с очень большим количеством процессоров можно достичь высокой эффективности.

Полное время выполнения параллельного алгоритма t_p является суммой времени, затрачиваемого на вычисление t_{comp}, и времени, затрачиваемого на коммуникации (обмен данными между процессорами) t_{comm}.

Время вычислений оценивается как время для последовательного алгоритма. $t_p = t_{comp} + t_{comm}$. Обозначим время запуска (startup), что иногда называют время скрытого состояния сообщений (message latency), как $t_{startup}$. Как начальную аппроксимацию *времени коммуникации* возьмем: $t_{comm} = t_{startup} + n * t_{data}$. будем считать $t_{startup}$ постоянным и зависимым как от оборудования, так и от программного обеспечения. Время передачи одного слова данных t_{data} также считать постоянным. Пусть от одного к другому процессора передаются *n* данных, *тогда теоретический время коммуникаций* мож представить в виде графика (див.рис.2.20).

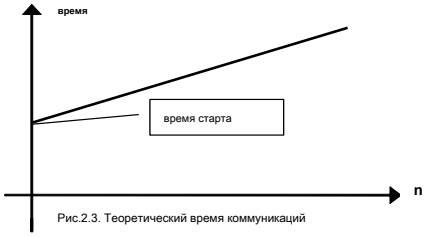


Рис.2.3. Теоретический время коммуникаций

для передачи q сообщений, каждое из которых имеет длину n-данных, необходимо время $t_{comm} = q\ (t_{startup} + nt_{data})$. Информативной величиной является также отношение вычислительных затрат к коммуникационным: t_{comp} / t_{comm}.

Определение для вариантов программы, имеющие различную величину

$(m)T_k$ продолжительность программы с величиной проблемы T и максимальным показателем распараллеливания kPc • на k процессорах.

Показатель масштабируемости некоторой проблемы, величина которой n на k процессорах сравнению с меньшей проблемой T (m <n) на одном процессоре определяется так:

если $T_1(m) = T_k(n)$, то есть продолжительность выполнения "малой" программы на одном процессоре равна продолжительности выполнения "большой" программы на k процессорах, то показатель масштабности можно выразить формулой:

$$\overline{SC_{mn}^{k}}$$

Заметим, что продолжительность выполнения зависит от такого параметра, как "величина проблемы", которая точно не определена. Ее в данном аспекте можно рассматривать как количество данных, которые обрабатываются вариантами программ различной величины по одному и тому же алгоритму.

На практике с увеличением количества процессоров чаще всего решаются большие проблемы и в большинстве практических применений не ставится задача решать те же проблемы быстрее за счет увеличения количества процессоров. В таких практических областях показатель *Scaleup* имеет большее значение, чем *Speedup.*

График зависимости показателя увеличения сложности решаемых задач SCк от количества процессоров приведен на рис.2.21.

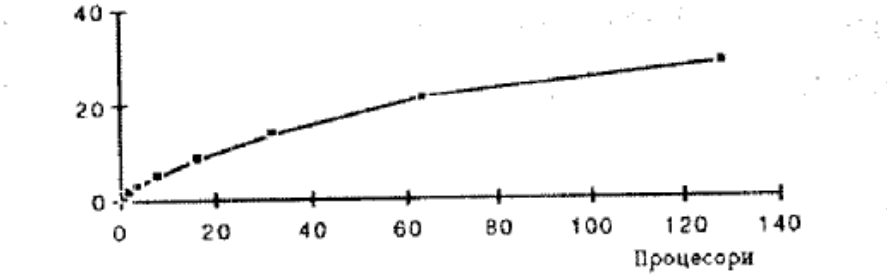


Рис.2.21 Зависимость показателя увеличения сложности решаемых задач SCк от количества процессоров

10. Сравнение MIMD и SIMD структур по производительности

Выше, при анализе производительности параллельных вычислительных систем ни различались MIMD- и SIMD-системы. Однако каждая программа включает в себя как минимум два различных максимальных показателя параллельности: один для асинхронной параллельной обработки, P_{MIMD}, а второй - для синхронной, P_{SIMD}. В связи с универсальностью асинхронной параллельной обработки имеет место соотношение $P_{SIMD} \bullet P_{MIMD}$.

На практике необходимо обратить внимание на следующие моменты:

- если в MIMD-системе есть свободные процессоры, которые не используются в данной задаче, то они могут быть применены другими пользователями для своих задач, решаемых одновременно. В SIMD-системах неактивизированном процессоры не используются;

- способ обработки информации в SIMD-программах дает существенно меньшие показатели разветвления;

- процессорные элементы SIMD- систем, как правило, менее мощные, чем процессоры, применяемые в MIMD-системах, но этот недостаток компенсируется за счет значительно большего числа процессоров SIMD- систем по сравнению с MIMD-систем.

В MIMD-системах часто программируются задачи в "SIMD-режиме", то есть не используется возможна независимость отдельных процессоров. Особенно это замечается, когда задача, решается, распределяется между большим количеством процессоров, то есть когда речь идет фактически о массивной параллельность. Это обстоятельство является одним из самых весомых аргументов в пользу SPMD- модели параллельности (same program, multiply data), которая возникает в результате смешанного использования SIMD и MIMD в рамках одной системы.

В связи с тем, что по закону Амдала эффективность сильно зависит от количества процессоров, а с другой стороны, не имеет возможности избежать некоторой последовательной части программы (как минимум, остаются программы ввода-вывода данных), возникает вопрос: целесообразно ли вообще использовать параллельную SIMD-систему с количеством процессоров?

Если ошибочно применить закон Амдала, то можно получить неверные результаты, к которым прежде всего приводит способ учета инструкций (рис.2.5). В то время как в программе А (например, в MIMD- программе) в соответствии с формулой Амдала учитывается каждая элементарная операция (это дает фактор f_A), в программе В для SIMD-системы как скалярные, так и векторные операции учитываются как одна инструкция (это дает фактор f_B). Такое определение является естественным для SIMD-системы, поскольку каждая операция, так учитывается, требует примерно одинакового времени на выполнение. Итак, имеем:

- f_A - последовательную часть программы относительно количества элементарных операций;

- f_B - последовательную часть программы относительно продолжительности выполнения операций. На рис.2.5 SIMD-программа может работать параллельно половину своего времени ($f_B = 0{,}5$).

В то же время количество последовательно выполняемых элементарных операций существенно меньше, а именно f_A = 1/6. Зависимость между этими двумя факторами выражается формулой

$$f_A \bullet \frac{f}{* (1 \bullet N) \bullet f f_{BBB}}.$$

Для получения точных данных производительности, можно применить для каждой SIMDинструкции выраженную в процентах количество активных в этой инструкции процессоров относительно их общего количества: это будет число между нулем (в случае выполнения скалярной операции в управляющей ЭВМ) и единицей (случай, когда все ПЭ активные) .

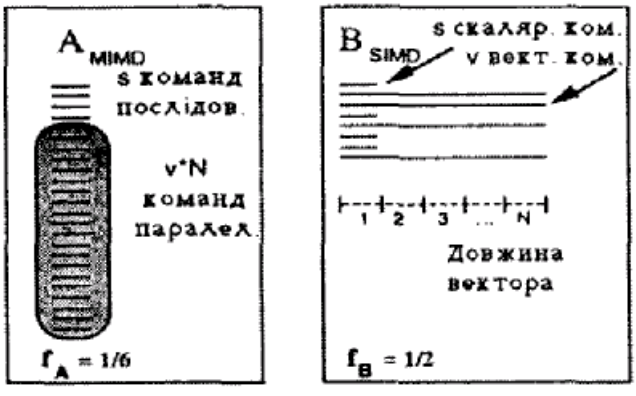


Рис. 2.22 Оценка количества параллельных инструкции

Для определения показателя ускорения SIMD-системы можно перечислить T_N, на T_1, в соответствии с вопроса: "Сколько времени потребовалось бы последовательной ЭВМ для выполнения этой параллельной программы?".

определение:

- f_B - часть скалярных инструкций (в процентах) SIMD-программы относительно общей количества (скалярных и векторных) инструкций.

Можно считать идентичным и такое определение

- f_B: часть времени выполнения последовательных операций (в процентах) относительно общей продолжительности выполнения параллельной программы.

тогда,

$$\bullet \quad * \quad \bullet \quad * \quad \bullet \quad * \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet$$

примеры применения измененного закона

1.

- Система имеет 1000 процессоров.
- Программа имеет максимальный показатель распараллеливания 1000.
- • 0,1% программы (в SIMD-исчислении) выполняется последовательно, то есть f = 1/1000.

Вычисление показателя ускорения:

$$S_{1000} = 0.001 + (1-0.001) * 1000 \bullet 999$$

В этом случае достигается ускорение, близкое к теоретически максимально возможного (E_{1000} = 100%).

Примерный результат получаем и по формуле Амдала с соответствующим f_A.

$$f_A \bullet \frac{f}{* \ (1 \bullet B \ ) \bullet} \bullet \frac{0.001}{1000 \ * \ 0.001 \ ) \bullet 0.} \bullet 6 \ .$$

Показатель ускорения по формуле Амдала:

$$S_{1000} \bullet \frac{1000}{* \ )} \bullet 999 \ .$$

2.

- Система имеет 1000 процессоров
- Программа имеет максимальный показатель распараллеливания 1000
- 10% программы (в SIMD-исчислении) выполняется последовательно, то есть $f_B = 0.1$.

Вычисление показателя ускорения:

$$S_{1000} = 0,1 + 0,9 * 1000 = 900.$$

Несмотря на заметно большую последовательную часть SIMD-программы достигается 90% максимального ускорения!

Приведенные здесь соображения направлены на четко очерченную проблему (с соответственно большим максимальным показателем распараллеливания) и на заданную неизменную количество процессорных элементов: ни размеры задачи, ни количество процессоров не изменяются, производится только сравнение с последовательной системой. В случае, когда меняется количество ПЭ в процессе выполнения программы, оценить, как меняется при этом показатель *speedup,* по приведенной формуле для S_N невозможно, так как она базируется на параметре $T_{N/}$ Увеличение количества ПЭ не привело бы к более быстрому выполнению той самой SIMD-программы в области $N > P_{SIMD}$ (количество ПЭ больше или равна максимальному показателю SIMD - распараллеливание), потому что эти дополнительные ПЭ были бы лишними и оставались бы неактивными. вычислив все T_i, где $1 < и < N$, можно найти соответствующие показатели ускорения S_i:

$$\bullet \quad \bullet \quad \bullet \ ; \qquad \bullet$$

Приведенная выше формула для S_N использована для экстраполяции продолжительности выполнения масштабируемых вариантов проблемы с линейным масштабным коэффициенты (*Skaleup)* и определена как " *scaled speedup* "(Показатель ускорения, связанный с масштабами задач), легко вводит в заблуждение: здесь исследовались временные показатели различных по масштабам вариантов программы и вычисленные показатели ускорения относительно этих вариантов. Было также принято, что последовательная часть (по формуле Амдала) некоторой "реальной" программы при ее масштабировании остается постоянной, а увеличивается только параллельная часть. Однако по определению это класс программ с линейным показателем масштабности (*scaleu* то есть для него справедлива формула

$$( )_N \bullet \quad p\ TAT(\quad * \ AK$$

где k - число, которое показывает, что в k раз больше вариант программы выполняется в k раз большим количеством процессоров за то же время.

Кроме того, всегда выполняется условие:

$_1(\quad * \quad) k \bullet A k T * \quad _1(AT)$ - программа, больше в k раз, требует при выполнении на одном процессоре в k раз больше времени.

С этого ограниченного класса задач, следует линейный прирост показателя " *scaled speedup* ":

$$\frac{* \quad (kS * A)}{{}^{k}{}_{NN}(AS)} \bullet \frac{\dfrac{(kT * A)}{{}_{*1} \ {}^{*}kT * A}}{\dfrac{{}_{k1}(AT)}{{}_{NN}(AT)}} \bullet \frac{\dfrac{* \ k \ (AT)}{{}_{*1} \ (kT * A)}}{\dfrac{{}_{k1}(AT)}{{}_{NN}(AT)}} \bullet \frac{* \ k_N (AT)}{{}^{*}kN \left(\dfrac{AkT}{} \right)} \bullet k .$$

ВЫВОДЫ

Все без исключения данные, характеризующие производительность параллельных вычислительных систем, должны рассматриваться критически. Для этого есть целый ряд причин.

1. Характеристики производительности, как показатель ускорения и показатель масштабности всегда связаны с конкретным применением параллельных систем - показатели справедливы только для данного класса задач и очень условно могут быть распространены на другие задачи.

2. Показатель ускорения некоторой программы касается только одного отдельного процессора параллельной системы.

3. В SIMD- системах процессорные элементы, как правило, имеют значительно меньшую мощность по сравнению с MIMD-процессорами, что может дать на порядок, а то и больше, разницу в оценках скорости.

4. Загрузка параллельных процессоров - главная сложность в MIMD-системах. В SIMD-системах это не так важно, потому что неактивные процессоры не могут быть использованы другими задачами. Несмотря на это, показатель ускорения исчисляется в зависимости от характеристик загрузки. Если SIMD-программа попытается "включить в работу" ненужные ПЭ, то данные загрузки, а с ними и показатели ускорения, станут недействительными. Параллельная программа в этом случае будет менее эффективной, чем по результатам тестирования.

5. Сравнивая параллельную систему (например, векторную) с последовательной (скалярной), нецелесообразно применять два раза один и тот же алгоритм (в параллельной и в последовательной версиях).

пример . OETS- алгоритм - типичный алгоритм сортировки SIMD-систем, но для последовательных процессоров эффективным является алгоритм Quicksort - во всяком случае Quicksort не может быть так просто переведен в эффективную параллельную программу для SIMD-систем. Сравнение "OETS- параллельного" и "OETS- последовательного" алгоритмов дает хорошие результаты для параллельной системы, но оно не имеет практического смысла!

Упражнения и задания к теме №2

1. Параллельная программа выполняется на MIMD - системе с 100 процессорами, 3% всех команд при проходе программы выполняются последовательно, а остальные может выполняться параллельно на всех процессорах. Какое значение имеет показатель ускорения этой программы на данной программе?

2. Некоторая параллельная программа, имеющая 10% последовательную часть, должен выполняться на MIMD - системе. Существует некоторое максимально возможный показатель ускорения, независимый от количества процессоров системы?

3. Параллельная программа имеет выполняться на MIMD - системе с 100 процессорами, однако:

- 2% всех команд при проходе программы должны выполняться последовательно;
- 20% всех команд могут выполняться только на 50 процессорах. Какое значение

имеет показатель ускорения для этой программы?

4. Параллельная программа имеет выполняться на SIMD-системе, имеющей 10000 процессорных элементов, однако она содержит при выполнении 20% скалярных команд. Остальные - векторные команды выполняются на всех ПЭ. Какое значение имеет показатель ускорения для этой программы?

5. Параллельная программа имеет выполняться на SIMD-системе, имеющей 10000 процессорных элементов. Если все ПЭ были активными в течение 30% общей продолжительности выполнения программы, а остальное время были неактивными, какое значение имеет показатель ускорения для этой программы?

6. Параллельная программа имеет выполняться на SIMD-системе, имеет 100 000 процессорных элементов, однако:

- 20% всех выполняемых инструкций является скалярными командами;
- 10% всех инструкций могут выполняться векторно только на 100 процессорах;
- 40% всех инструкций могут выполняться векторно на 50000 процессорах;
- остальные инструкций могут выполняться векторно на всех процессорах. Какое значение

имеет показатель ускорения для этой программы?

7. Можно ли для оценки производительности процессора с плавающей точкой использовать единицу измерения MIPS?

8. Почему для оценки производительности параллельных систем неэффективный метод вычисления производительности составных частей?

9. Недостатки относительно вычисления производительности имеет закон Амдала?

10. Как влияют параметры коммуникаций на общую производительность систем различного типа?