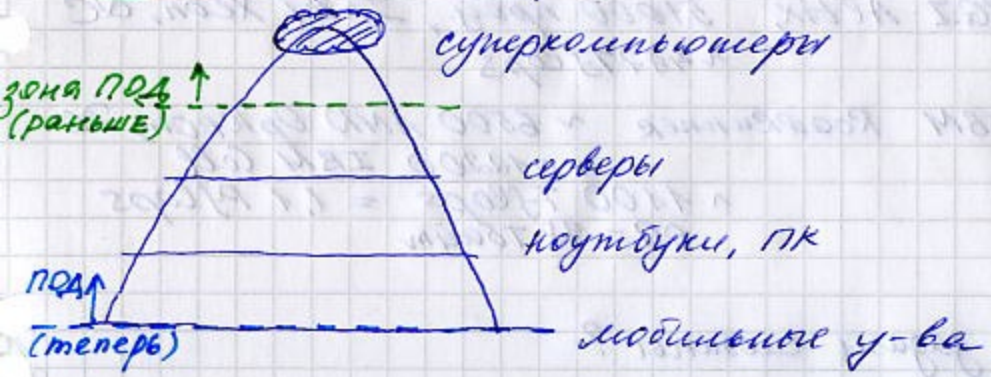
Параллельная обработка данных

8.09

Компьютерный мир



→ Смерча 1956
300 м²
150 кВт
2000 оп/сек

Совр. ПК
~ 10 Gflops

flops = floating point operation per second

- веществ.
- арифмет.
- плав. запятая

M - 10^6, G - 10^9, T - 10^{12} P - 10^{15}

~ 1-2 GBytes
~ 400 GB (жест. диск)

Top500.org список самых мощных

Cray XT4, ~36000 проц., AMD Opteron, DC
~266 Tflops

IBM Blue Gene, 212990 процессоров
IBM PowerPC 440
~ 478 T-flops

SGI Altix, 51000 проц., Intel Xeon, QC
~ 487 T-flops

- IBM Roadrunner ~ 6500 AMD Opteron, DC
+ ~ 12200 IBM Cell
~ 1100 T-flops = 1,1 P-flops
ОП = 98 Тбайт

– задачи сложны?

– задачи важны?

задача: посчитать кол-во счастливых билетов

```
for (i0=0; i0<=9; ++i0)
  for (i1=0; i1<=9; ++i1)
    ...                    // i2
     ...                   // i3
      ...                  // i4
       ...                 // i5
        if (i0+i1+i2 ==
            i3+i4+i5)
          ++count;
```

10 цифр ~ 40 сек
12 цифр ~ 1,5 часа
14 цифр ноутбук делается
16 цифр – не доживется

10^N, N – число цифр

Даже суперк с нашим алгоритмом много не посчитает

∃ алгоритм сложности N^2

• Задача N ферзей
8×8 – быстро
25×25 – рекорд
14×14 10 ферзей – уже тяжело

Моделирование нефтяных резервуаров

Будем считать, что резервуар – прямоуг. параллелепипед

~ 100 точек

5-20 функций в 1 точке

Решить нелин. ур-е для 1 ф-ции, т.е. выполнить 200-1000 операций
~ 100-1000 шагов по времени

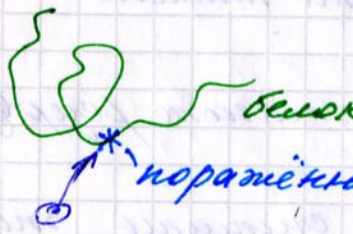
Всего операций: 10^6(точек) * 10(ф-ций) * 500(операций) * 500(шагов по времени) = $= 2,5 \cdot 10^{12}$

– автомобилестроение
кран-тесты
Формула 1: BMW в 2006 – кр на 12 T-flops
Williams в 2007 – суперк на 7 T-flops
Renault, 2008 36 T-flops

- авиадвигатели

ТОР50 – суперк. в СНГ supercomputers.ru

- проектирование лекарств



Надо подобрать лекарство

На проверку 1 соединения уходит от 1 мин до 10 часов

- кинопроизводство

Ирен 3 ~ 120000 кадров

1 кадр ~ 2 часа на 1 процессоре

$\sim 20 \cdot 10^6$ процессоро-часов

Суперк 8 тыс. процессоров

- суперкомпьютеры и наука
  - теория
  - эксперимент
  - вычислит. наука
- whirpool
- Procter & Gamble

... жиры

EDSAC, 1949г 15.09

такт $2 \cdot 10^{-6}$ с

произв-ть 100 оп/с

IBM BlueGene 2007-2008

такт $1,4 \cdot 10^{-9}$ с (700 MHz)

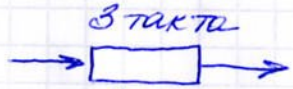
произв-ть $4,78 \cdot 10^{14}$ оп/с

~1500 раз

$4,7 \cdot 10^{12}$ раз

Принципы ПОД

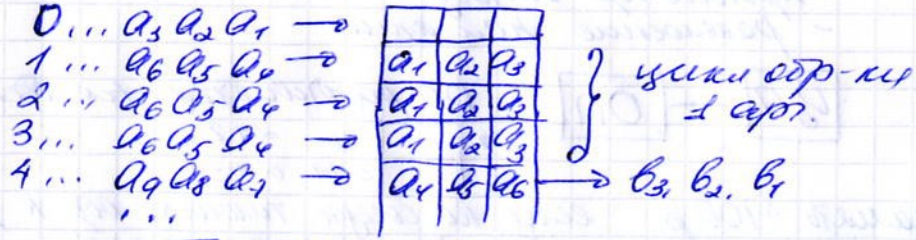
- параллельность
- конвейерность



T = 900 тактов

параллельная обработка – N устр-в

в N раз быстрее



T = 300 тактов

Конвейерная обработка

⇒ [| | | |]

микрооперации (например, при сложении вещ. чисел: сравнение порядков, выравн. порядков, сложение и т.п.)

0... $a_3 a_2 a_1$ [| |] 3 ступени конвейера, каждая 1 такт

1... $a_4 a_3 a_2$ → [a_1 | |]

2... $a_5 a_4 a_3$ → [a_2 | a_1 |] — цикл обработки 1 арг-та

3... $a_6 a_5 a_4$ → [a_3 | a_2 | a_1]

4... $a_7 a_6 a_5$ → [a_4 | a_3 | a_2] → b_1

... → $b_2 b_1$

$T = 3 + 299 = 302$ такта

Экскурс в историю

1) IBM 701 (1953г), IBM 704 (1955)
разр-парал — память
— арифметика
ферритовая память

2) IBM 709 (1958)
независимый ввод/вывод

3) IBM STRETCH (1961)
- просмотр вперёд
- расслоение памяти

[ЦП] — [ОП]   ОП разбивается на банки

α $\alpha+2$
$\alpha+1$ $\alpha+3$

память ~100 р. медленнее (и по сей день)
если на соседн. тактах обр. к разным банкам, обр-ка параллельная

4) ATLAS (1963)
конвейер команд

→ [ ] → 6 тактов

[| | |] 4 ступени
выборка адреса операнда, исполнение и т.д.

(1,5) раза, получим (~1,6) р.

ветвление:

накладн. расходы

5) CDC 6600 (1964г)
- 10 независ. функц. устр-в

такт 100 нс
произв-ть 2-3 млн оп/сек

6) CDC 7600 (1969г)
- 8 независ. конвейерных ФУ

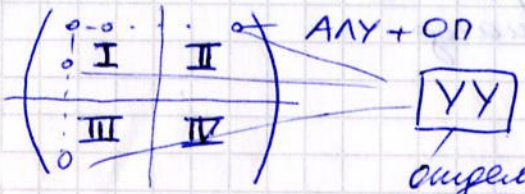
такт 27 нс
произв-ть 10-15 млн оп/сек

- пиковая произв-ть: (недостижима)
$\frac{\text{число опер-й}}{\text{время}}$

- реальная произв-ть (опр. программой)
всегда намного меньше пиковой

7) ILLIAC IV (1974)
- матрица процессоров

16×16 – 256 процессоров



отдельно
одно на всех

в 4 момент вр. выдаёт всем команду

Процессоры разделены на 4 квадранта.

Проект: 40 нс, 256 ПЭ
~1 Gflops – пиковая произв-ть

1967 – начало проекта
1971 – квадрант (8×8)
1974 – сдача в эксп.

1/4, 80 н/с
реальная произв-ть 50 Mflops

Иерархия памяти

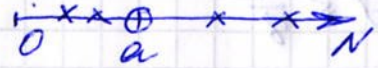
регистры
кэш-память
оперативная память
диски
ленты
...

скорость, $

– локальность вычислений

маш. код

– локальность исп-я данных



– 6 регистров в Прц. элементе
– ОП в ПЭ (2048 слов)
– 2 диска по 1 Gbit
– лазерный диск 10^{12} bit

22.09

Закон Амдала

ускорение в p раз (# процессоров)

$0 \le f \le 1$ – доля операций, кот. невозможно распараллелить

$$\frac{T_1}{T_p} = S \le \frac{1}{f + \frac{1-f}{p}}$$ ускорение ($\frac{t \text{ посл}}{t \text{ прц}}$)

$f = 0{,}1$, $p \to \infty$ $\Rightarrow$ $S \le 10$

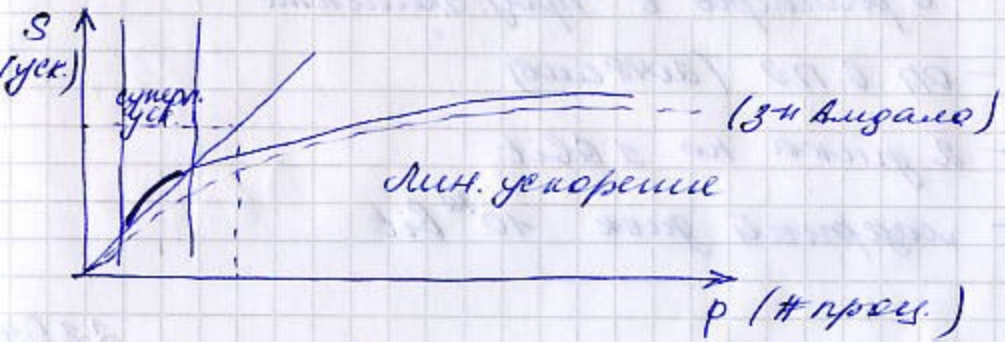
$\Rightarrow$ не смотря на большое # прц-в не получим ускорение > 10 раз

Следствие: для ускор. пр-ма в q раз необходимо ускорить в q раз не менее чем $(1 - \frac{1}{q})$-ю часть программы. ⑤

$q = 100 \Rightarrow \left(1 - \frac{1}{q}\right) = 99{,}99\%$ пр-ма

$\Rightarrow S = q$ необх, чтоб $f \le \frac{1}{q}$

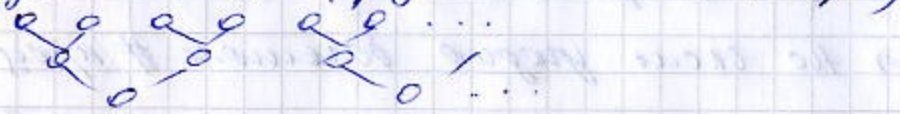
Суперлинейное ускорение



при эффективном использовании кэш-памяти можно получить больше, чем лин. ускорение.

```
S=0;
for(int i=0; i<n; i++)
    S=S+A[i];
```

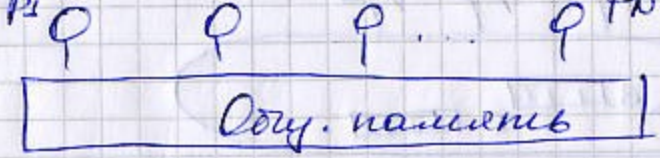
Сдваивание (разбиваем A на пары)



проблемы: – разные вычислительные способности проц-в
– особенности массива A
– задержки при переключении контекста
...

Архитектура вычислителя

1) Комп. с общей памятью



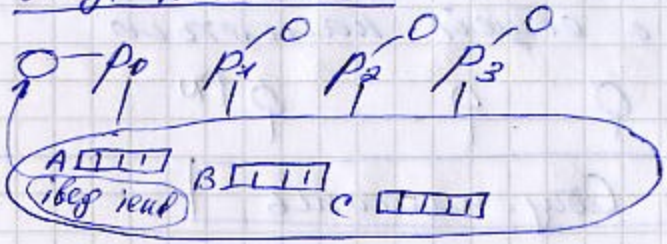
SMP (Shared Memory Processors)
Symmetric Multi Processors

2) Комп. с распред. памятью

„Узкое место" – канал связи с ОП
№564

```
N=1000
for(i=0; i<N; ++i)
    Ai = Bi + Ci * x;
for(i=0; i<N; ++i)
    Ai = (Ai + Ci) + BN-i-1;
```

Общ. память

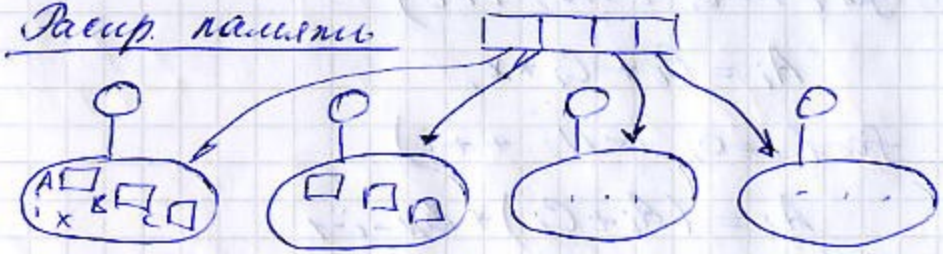



```
N = 1000;
n-pr = 4;
size = N/n-pr;

ibeg = proc_id * size;
iend = proc_id * ibeg + size;

for (i=ibeg; i<iend; ++i)
...
for (i=ibeg; i<iend; ++i) ...
```

Классы переменных

глобальные: A, B, C, x
локальные: i, ibeg, iend

Распр. память



$N = 1000/4 = 250$

<обмен 1>

```
for (i=0; i<N; i++)
...
```

<обмен 2> В

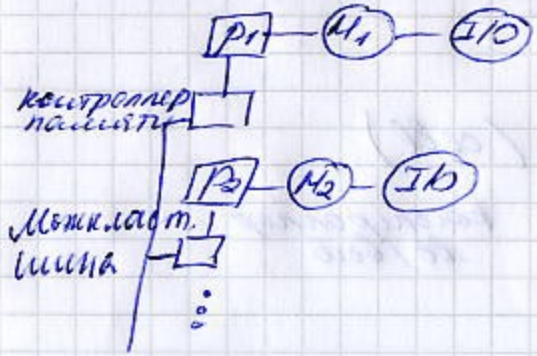

```
for (...)
```

<обмен 3> А

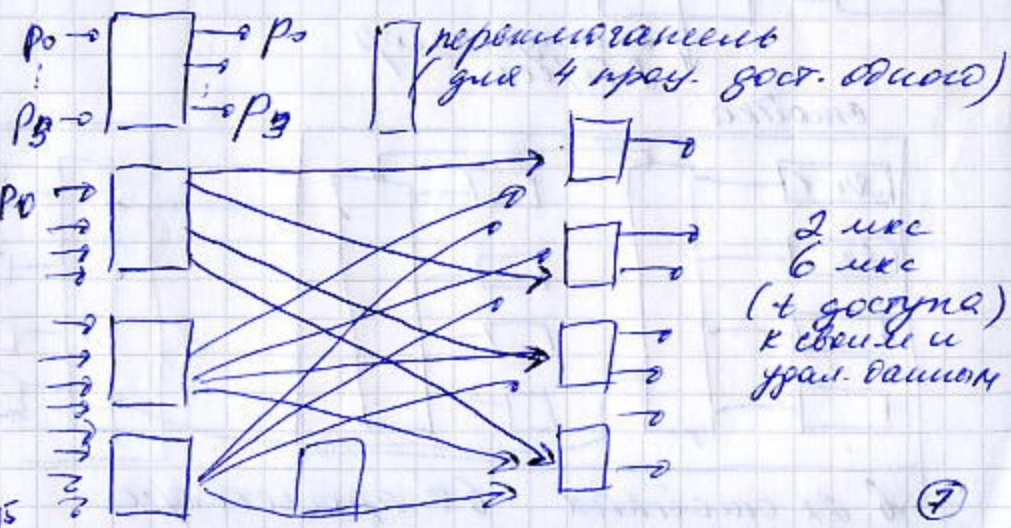
К-ри класса UMA (Unified Memory Access)

NUMA (Non-UMA)

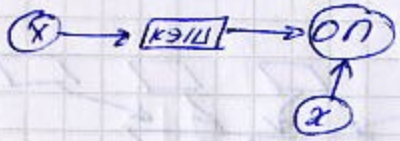
Впервые – в комп. Cm*



BBN Butterfly



ccNuma
cash coherent



HP SuperDome (2000 г)

до 64 проц-в

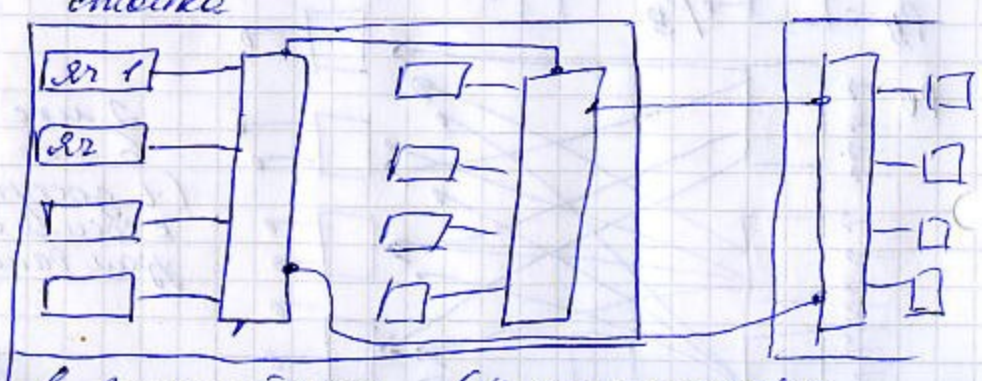
ОП ≤ 256 Гб - 1 Тб

PA 8600 / 8700

Itanium

вычислит. ячейка (cell)

стойка



в 2х стойках 64 процессора

PA 8700

750 МГц

4 фл / такт

пик. 3 Гфлопс

×64

192 Гфлопс

Вычислит. пиковой произв-ти: 6.1

Механизм произв-ти:

1) Закон Амдала

Ускорение $S < \frac{1}{\alpha + \frac{1-\alpha}{p}}$

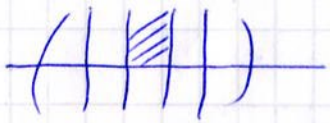
всегда есть последовательная часть

2) арх-ра ccNUMA обращение к памяти

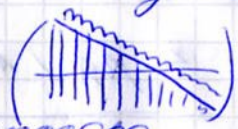
3) ccNUMA кэш

4) Равномерность распределения нагрузки между процессорами

⇒) В Сбербанке - 9 машин HP SuperDome (самая большая инсталляция)



Δ и-уда ⇒ ускор. деление



5) Производительность процессора

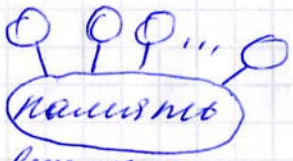
PA-8700
Itanium II

$\frac{\text{Non}}{T}$ $\frac{\text{реальная произв-ть}}{\text{пиковая произв-ть}}$ 15%

⇒ Эффективность работы 1-3%

6) ...

Компьютеры с распределённой памятью



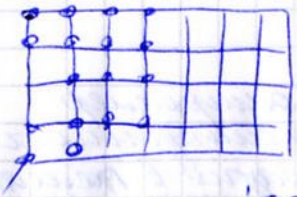
много времени – на обмен сообщениями

Начало 90х

- процессоры (какие?)
- коммуникационная сеть (как устроена?)

Intel Paragon

Intel Paragon

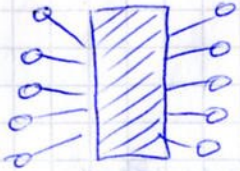


прямоуг. решётка

связь между соседями

процессоры i860

IBM SP1/SP2



связь от ∀ к ∀

комм-р – t уст-я связи невелико

Массивно-параллельные компьютеры

CRAY 1993-1994

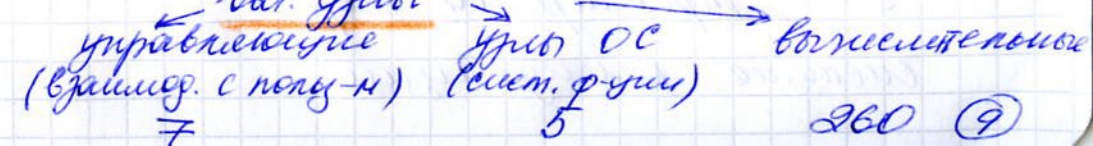
T3D, T3E, ~~T3X~~ не выпустили

XT3, XT4, XT5 (2 место в Top500)
в наст. вр.

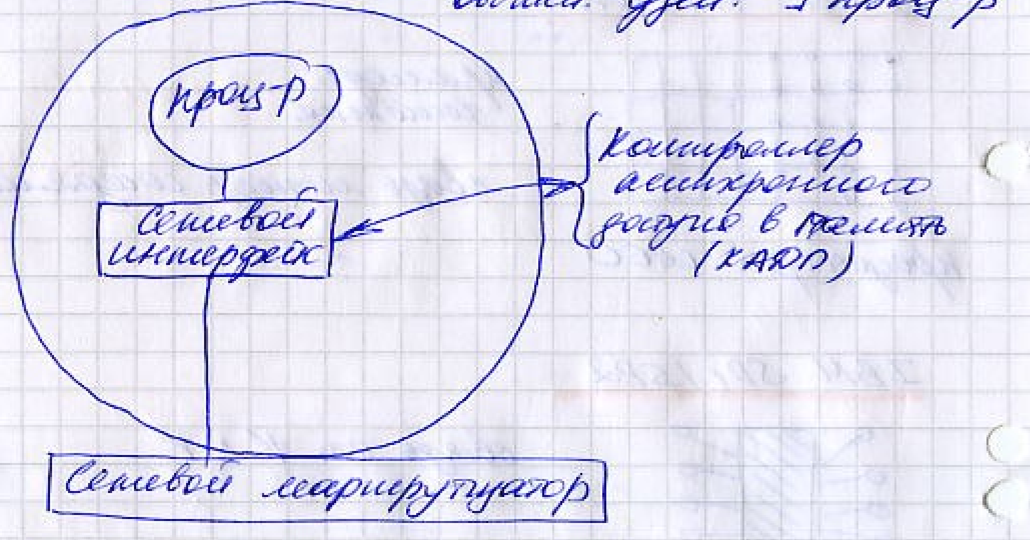
T3D, T3E – на осн. проц. DEC ALPHA (128)

2 проц. на узел XT3, XT4, XT5 – AMD Opteron (4 ядра)

- выч. узел
- коммуник. сеть

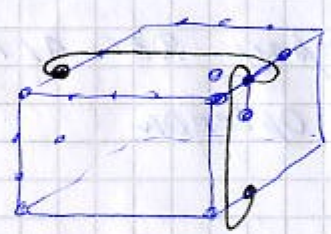


вычисл. узел: 1 проц-р

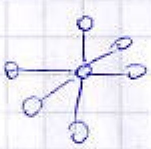


С помощью КАДП узел получает доступ к памяти др. узла, не нарушая его работу — быстрое общение процессоров

Комм. сеть: узлы решетки



Целочисл. точки внутри тоже узлы



у ∀ узла 6 соседей

„Трёхмерный тор" — система коммутации

повреждённые связи можно менять на обходной маршрут

за 3 перемещения можно попасть из 1 узла тора в другой

x_A, y_A, z_A — коорд. A

x_B, y_B, z_B — коорд. B

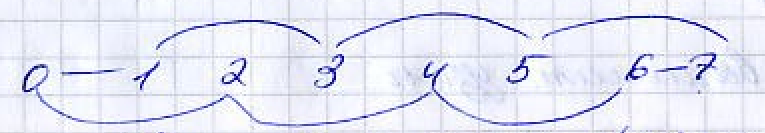
Маршрут из A в B:

$\overset{A}{x_A, y_A, z_A} \to \overset{A'}{x_B, y_A, z_A} \to \overset{A''}{x_B, y_B, z_A} \to B$

Но ∀ тоже — кратчайший путь

A → B и B → A отличаются

длина проводника влияет

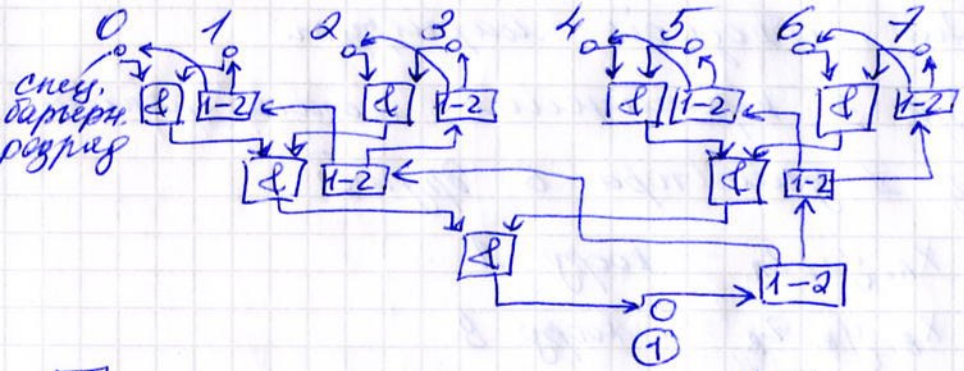


связи более-менее равномерны

Барьерная синхронизация

Суммарное кол-во сообщений — которые посылают всем (для синхронизации)

Краевое аппаратное решение:



[4] можно заменить на [V]

в гараф поиска в БД, например (до первой находки)

13.10

Вычислительные кластеры

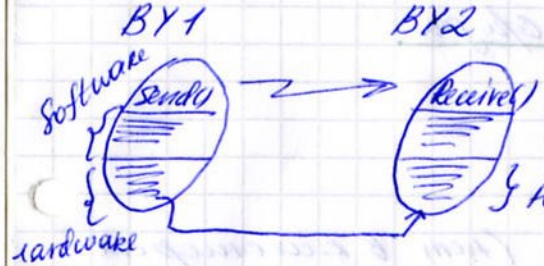
1994

- вычислит. узлы
- сетевая технология
- Linux, GNU, MPI

Cray XT5 $\frac{\text{цена}}{\text{произв-ть}}$ << для кластеров

Узлы можно подбирать под задачу

Узлы
- ОП → max
- макс. частота → max
- ОП, +HDD
- ускорители (Cell, GPU, ...)

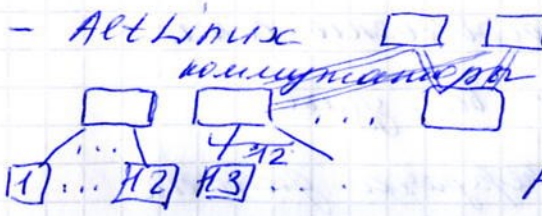


- латентность - время для пересылки сообщения нулевой длины (задержка)
- скорость передачи по сети
- цена

GE (gigabit ethernet)	InfiniBand
1 GB/s - скорость	20-40 GB/s
80-80 мкс - латентность	1-3 мкс
5 $ узел - цена	1000 $ узел

СКИФ МГУ "Чебышёв"

- вычисл. узлов: Intel Xeon 5345 (Clovertown), 2.3 GHz (625)
  (8 GB ОП, без дисков,
  8 GB ОП, HDD
  16 GB ОП, HDD
  32 GB ОП, HDD) 8 штук
- InfiniBand DDR
  GEthernet (для поддержки работы ОС)
  ServNet (для мониторинга узлов)
- AltLinux



Вычислительные среды

- массовость
- удалённость
- динамичность (нет в кластерах и CRAY/XT5...)
- неоднородность — " —
- различная административная принадлежность (разные налог. и др. политики и т.д.)

grid - технологии - все предоставляют собой единую среду (метакомпьютинг)

cloud computing - облачные вычисления

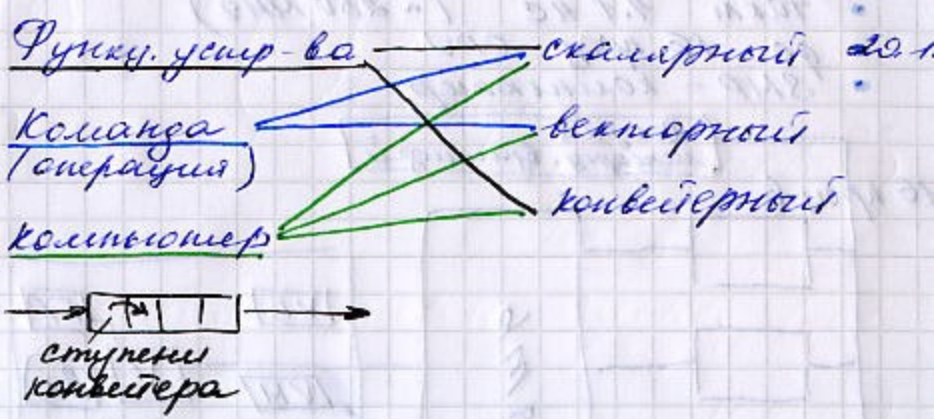
Идея:

Из сущ.-х ресурсов формир. среда, дающая сервис высокого уровня

Проблемы: (накладные расходы)

1. Закон Амдала $S < \frac{1}{\alpha + \frac{1-\alpha}{n}}$
2. Латентность
3. Пропускная способность сети
4. Баланс загрузки вычисл. узлов
5. Асинхронная передача данных

6. Реальная произв-ть процессоров



$A: x, y \to t$ - скаляр

$A: V_1, x \to V_2$ - вектор

Векторно - конвейерные компьютеры

1976 - Cray-1

начало 90х Cray C90 (T90)

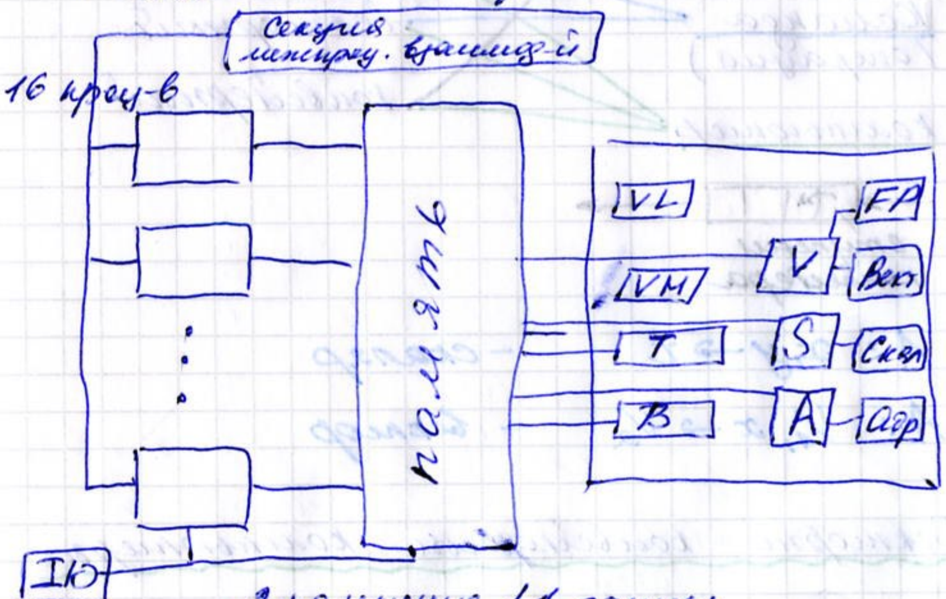
$\frac{\text{цена}}{\text{произв-ть}}$ велико

пиковая пр.
| считается разница
реальная пр.

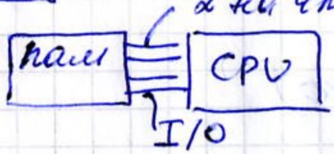
2002 NEC "Earth Simulator" 40 Tflops
(5 редакций TOP500)

CRAY C90

- такт 4.1 нс (~250 МГц)
- до 16 проц. CPU
- SMP – компьютер



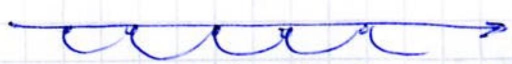
2 на чтение / 1 запись по кр. мере



память

0 0с 0п 0б
1 1с 0п 0б
...
7 7с 0п 0б
8 0с 1п 0б
...
63 7с 7п 0б
64 0с 0п 1б
65 1с 0п 1б
...

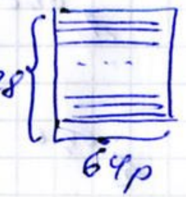
макс. задержка 7 тактов при обр. к с. и подс. одинаковым – с шагом 64



$h = 2^k \cdot l^{нч}$ k=6 – макс. конфл.
k=0 – мин. конфл.

- регистры
  - Осн. группа
    - адресные 8 шт 32 разр
  - Вспомогат. группа
    - скалярные 8 шт 64 разр
    - векторные

Вект. рег. – совок. из 128 регистров 64 разр.
таких 8 штук



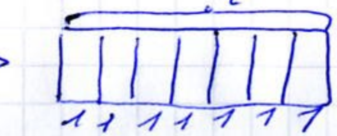
Вспомогат.:
- 64 шт. 32р (B) (для A)
- 64 шт. 64р (T) (для S)

VL – регистр длины вектора (8р)

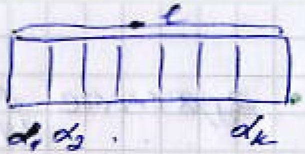
VM – регистр маски

```
if (...)   Ai = ...
else       Ai = ...
```

- Функц. устройства
  - независимы
  - конвейерные
  - ∀ ступень – 1 такт

(n) ⇒ 

(l+n-1) время в тактах для масс. из n эл-тов



$max(d_k)$ — частота получения результата

1. Адресные ФУ

   2 шт., 32 разр., скал. операции. $A_I^{32,c}$

2. Скалярные ФУ

   4 шт., 64 разр., скал. опер., $A_I^{64,c}$

3. Векторные ФУ

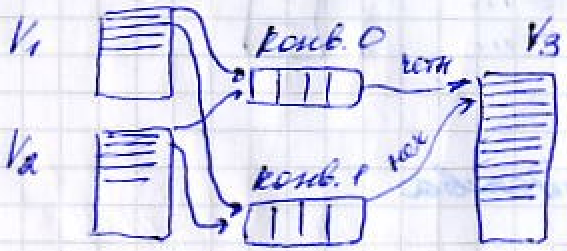
   5-7 шт., 64 разр. вект. опер. $A_I^{64,в}$

4. Вещ. арифметика — FP

   3 шт., 64 разр. скал/вект. $A_{FP}^{64,c}$ $A_{FP}^{64,в}$

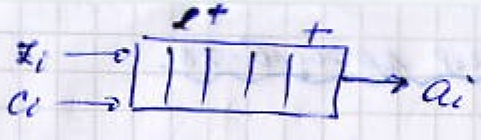
   $\pm, *, \frac{1}{x}$

A: $V_1, V_2 \rightarrow V_3$



чётн. эл-ты на конв. 0
нечётн — на конв. 1

$a_i = b_i * x + c_i$



$(l^* + n - 1 + l^+ + n - 1)$ тактов на вып.

Зацепление ФУ: z_i сразу идет на l^+: $(l^* + l^+ + n - 1)$ — ускорение в 2 раза на уровне архитектуры

Особенности, помогающие считать быстро:

— ФУ независимы
— ФУ конвейерные
— векторная обработка

```
for (i=0; i<n; ++i)
    A[i] = B[i] + C[i];
```

L: $B \rightarrow V_1$
L: $C \rightarrow V_2$
A: $V_1 + V_2 \rightarrow V_3$
ST: $V_3 \rightarrow A$
} эффективнее в 10-20 раз

— дублирование конвейеров
— зацепление ФУ
— 16 процессоров

flop/s - числовая произв-ть

$\pm$, *, $\frac{x}{\sqrt{x}}$ забудем, т.к. редко исп-ся

Пик до 1 такт: 4 операции (2+, 2*)

4.1 нс => на 1 проц. 1 Gflops
(10^9 вещ. оп/сек)

27.10 2002 г - в Японии NEC "Earth Simulator"
40 Tflops / 2003 - 2005 #1 в Top 500
640 узлов * 8 процессоров
1 проц. ~ 8 Gflops

6 ФУ, 72 вект. регистра * 256 элементов

500 MHz
8 секций

SSE - инструкции - в Intel процессорах, исп. идею векторной обработки

Сегодня вект-конв. машины производят NEC и CRAY (прогноз погоды)

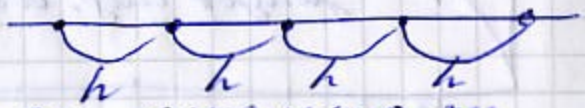
Векторизация программ

```
for (i=0; i<n; i++)
  A[i] = B[i] + C[i];
```

A^V; B, C → A
векторизация

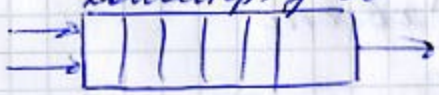
Условия: - вектора данных
- одинаковые правильные операции

вектор данных:



Один и тот же тип, одно и то же расст. в памяти

конвейер. у-во



```
for (i=0; i<n; ++i)
  A[i] = A[i-1] + B[i]
```

! конвейер работать не будет, т.к. операции зависимы

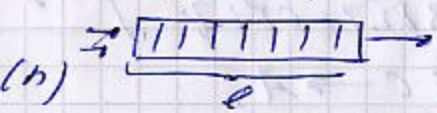
Проблемные места:

1. Закон Амдала

Ускорение $S \le \frac{1}{\alpha + \frac{1-\alpha}{p}}$

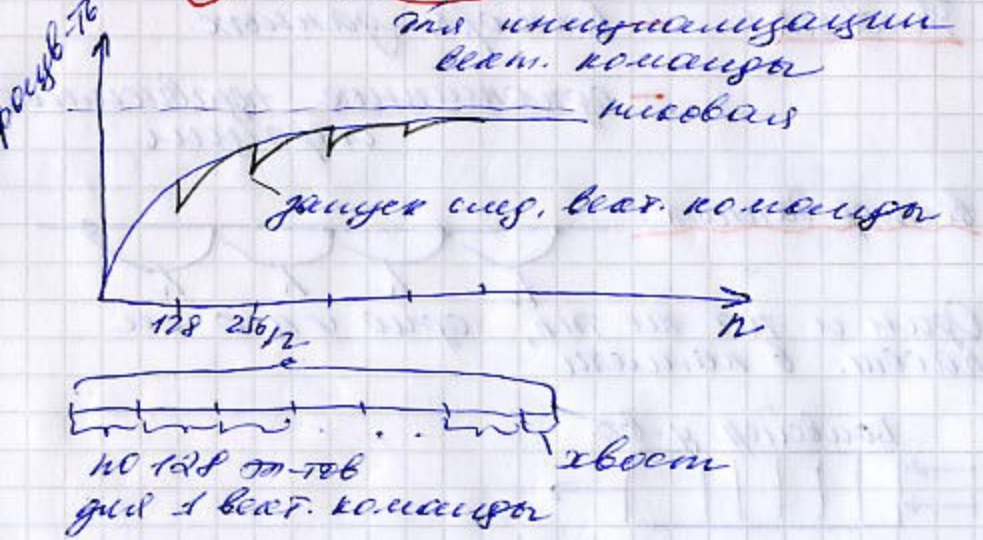
2. Время работы конвейера

3 Секционирование векторных команд



$T = l + n - 1 + d$

d – для инициализации вект. команды



Операция $a_i = b_i * 5 + c_i$

n	M-flops
1	7
4	27
64	300
128	430
129	364
256	548
257	490
...	...
8200	800

4. Конфликты в памяти

```
for (i=0; i<n*k; i+=k)
    a_i = b_i * 5 + c_i;
```

n = 1000

k	M-flops
1	405
2	444
4	294
8	142
16	84
32	44
64	22

```
do i=1,n
  do j=1,n
    do k=1,n
      X(i,j,k) = X(i,j,k) + P(i,j) * Y(i,j)
```

$X(i,j,k) \sim X(i-1,j,k)$ 1 $(X(b_1, b_2, b_3))$

$x(i,j,k) \sim X(i,j-1,k)$ b_1

$x(i,j,k) \sim X(i,j,k-1)$ $b_1 * b_2$

$X(40,40,\ldots)$ => расст. $1600 = 64 \cdot 25$

=> при векторизации эл-ты расположены худшим образом

Решение: $X(41,41,\ldots)$

* понятно – не будет конфл. в памяти

Адресация: $A(IA(I))$

$IA(I) = I$

$IA(I) = cons\,t$ – конфликт, одно и то же слово и

5. Ограниченная пропускная способность канала процессор-память

$a_i = b_i + s_i + c_i$

вт×   в×   в×   в×   - только 2 входа

6. Необходимость исп-я векторных регистров

7. Сбалансированность +/*

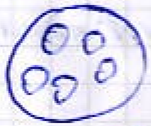
8. ...

A = B + C   вектора

A = B + C*1   произв. возрастет в 2 раз.

$\frac{N_{оп}}{T}$

Параллелизм на уровне машинных команд



сколько-то ФУ

- суперскалярная архитектура
- арх-ра с большим командным словом (VLIW)

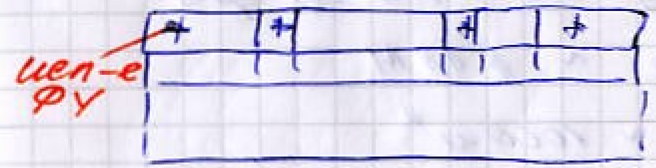
} - машинные инструкции

в процессе исполнения происходит распараллеливание

- почти во всех совр. процессорах

VLIW (Very Large Instruction Word)

Распарал. регламентируется компилятором



Все решается в статике - компилир. оптимальн. от исх. текста

GP (AMD) (графические)

эффективность GPU ←→ удобство и скорость программир. классические

FPGA   Cell   векторные   многоядерные

программир. логич. интегр. схемы (ПЛИС)

⇓

SMP-компьютер (общ. память) (эффект. + относит. удобно программир.)

⇓

К-р с распред. памятью

NEC SX-9

CRAY XT5(h)
- AMD (6-ядерные)
- векторные
- FPGA

⇓

Распред. вычислит. среды

3 11 Технологии параллельного программирования

Типовая кластер-ть 1.2 Pflops

Стоимость ~ $88M

Площадь ~ 1000 м²

Вес 150 т

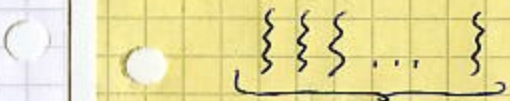
6000+ Intel

5000+ GPU (AMD)

Нужно, чтобы задача быстро решалась на компьютере

Модель SPMD

(Single Program - Multiple Data)



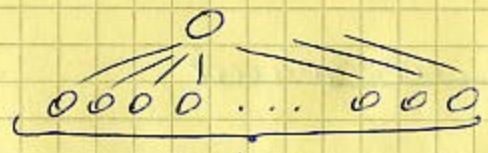
n процессов, у + свои данные

Как "подсунуть" разные данные?

Есть спец. функция типа my-id()

```
if (my-id()==3) { ... }
else { ... }
```

Master / Slaves (Workers)



n

процессы „общаются" только с мастером, кот. контролир. ситуацию

у мастера своя программа,

у рабов - своя

MIMD

1. Традиц. Я Прогр-я (Фортран, Си, ...)
2. Спецкомментарии

Open MP

pragma OMP

средство добавить комментарии к у-ву алгоритма

3. Расширения яп

HPF (High peoformance Fortran)

FORALL

mpC (расширение C)

Ограничение: вопрос к компилятору: для каких платформ он существует?

4. Новые языки

Occam

Sisal (язык функционального программирования)

Fortress

5. Системы передачи сообщений

MPI
PVM
Shmem (Shared memory, функц. CRAY, активная синхрония одна)

6. Параллельные библиотеки

LAPACK (лин. алгебра)

FFTW

MKL (от Intel)

7. Специализированные пакеты

ANSYS, CFX, FlowVision
GAUSSIAN, GAMESS

НПО Сапурн используют штатные пакеты, они очень дорогие для коммерческого использования

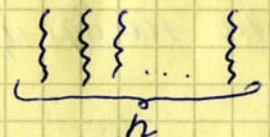
Много времени проходит между новой технологией и внедрением => в пакетах устаревшие модели

Решение: разработка программ с открытым кодом

Нужно найти компромисс между

Эффективностью пр-я, продуктивностью и переносимостью

Технология прогр-я Linda



пространство кортежей

Нет прямого общения процессов между собой, только через пр-во кортежей

кортеж: (17, true, 3.7)

("Винта", 28.1, 32.5, 0, 0)

Функции:

1. Out ("x", 12)

кортеж будет записан в пр-во кортежей, если такой уже был, добавится еще один
out никогда не блокируется

2. **In**("y", 28, 13)
забрать ту пр-во кортежей указанного
можно в кач-ве пар-ра указать шаблон:

In("имя", int i, 13)
какое-то целое, если несколько кортежей подходят, случайно выбир. один

In("имя", ?i, 13)
если перем. i уже определена в пр-ме (разная область видимости)

In("имя", i, 13)
выбрас. значение i и переименовается

In("x", ?i, ?j, int k)

Если ни один кортеж не найден, процесс блокируется

3. **Read**(— " —)
не удаляет кортеж ту пр-ва, а только читает данные

4. **eval**("p", 17, g(x)+12, f(x)*f(x));

идёт дальше не дожидаясь заверш. работ процессов

порождается параллельный процесс
a=g(x)+12

порожд. параллельный процесс
b=f(x)+f(x)

→ ("p", 17, 9, 6) ⟶ в пр-во кортежей

out("Next", 0); "эстафетная палочка"

{ In("Next", ?my_id);
{ Out("Next", my_id+1);

все процессы себя перенумеруют

read("Next", ?numbers)
общее # процессов

Out("Barrier", n);

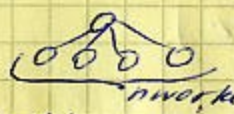
In("Barrier", ?b);

b=b-1; // не явл-ся ли процесс последним?

if(b!=0) { out("Barrier", b);
read("Go"); // блокировка, пока не появится кортеж "Go"
}

out("Go");

порождение workers



for(i=0; i<nworker; ++i)
eval("go", worker(i));

+ параллельная программа может быть написана в терминах Linda
- ограничение размера пр-ва кортежей
- общая память (для мультипроцессоров и с/м с распр. памятью надо моделировать)
- поиск нужного кортежа

10.11

Оценка производительности параллельных вычислит. систем

— Пиковая произв-ть

- MIPS, млн. команд/с (миллион оп/сек)

C/C++/F

≡

486 — (487) — a+b

процессор сопроцессор

≡ } a+b

в терминах MIPS в прогр. характеристика лучше, а фу. времени меньше у сопроцессора

- FLOPS (вещ. оп./сек)

a+b ⟷ ≡ +
1 к 1

— тест Linpack - основа top500

решение СЛАУ с плотной м-цей (LU-разложение)

1. м-ца 100×100 запрет на изменение текста программы
2. размер 1000×1000 [— инициализация
— → решение СЛАУ
— $\frac{2}{3}n^3 + 2n^2$ (м.б. полувой операции)]
3. Размер м-цы $N_{max} + N_{max}$ - которые „влезают" в вычисл. систему (разрешено исп-я BLAS - базовую библиотеку лин. алгебры)

Всем наш разработ далек от СЛАУ, тест не очень подходит

Top 500
R_{peak} - пиковая
R_{max} - на Linpack

$\frac{R_{max}}{R_{peak}} \cdot 100\% \sim$ КПД

N_{max}

$N_{\frac{1}{2}} \sim \frac{R_{max}}{2}$

— Perfect Club Benchmarks

$\Sigma \sim 50\,000$ строк

Много приложений из разных областей

Сложно перенести тест для разных платформ - от теста вскоре отказались

- NAS Parallel Benchmarks
  - 5 вычислит. ядер (FFT, IS, CG, ..)
  - 3 модельных приложения
  NPB 1.0 – фикс. задача
  NPB 2.0 – можно только опции компилятора; допускается внесение изменений < 5% исх. текста
  A, B, C (D) – класс памяти (объём)

HPC Challenge

- HPL (Linpack)
- умножение м-ц
- транспонир. матриц
- FFT
- Random Access (работа с памятью, все проходят плохо)
- латентность / скорость передачи данных по коммуникац. сети
- STREAM
$$\begin{cases} a_i = b_i & \text{(пересылка)} \\ a_i = q * b_i \\ a_i = b_i + c_i \\ a_i = b_i * s + c_i \end{cases}$$

$S_{передачи}$ (скорости) $= \frac{R_{peak}}{S_{передачи}} > 1$ для почти всех с-м

Open MP

C/C++, Fortran (77, 90, ..)

- спецкомментарии
- функции
- доступ к переменным среды

SPMD (Single Program, Multiple Data)

#pragma omp ⌙

!$ omp ⌙ // Fortran

```
!$OMP PARALLEL
   <парал. секция>
!$OMP END PARALLEL
```

OMP_NUM_THREADS – перем. среды

OMP_DYNAMIC – можно ли менять ↑ в процессе программы

OMP_NESTED – разрешение вложенного параллелизма

<условие> - вход в парал. секцию при выполнении условия

Распределение работы

1. Ф-ции нижнего уровня

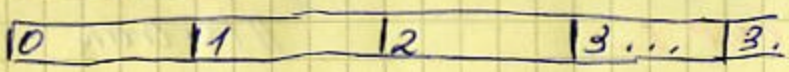
```
IF( get_omp_num_thread( ). EQ. 4) THEN
      <для тима 4>
ELSE
      <для остальных>
ENDIF
```

2. Парал. циклы

```
!$ OMP DO <распр-е итераций>
```

а) STATIC [, m] - блочно-циклическое
необяз. пар-р (размер блока)

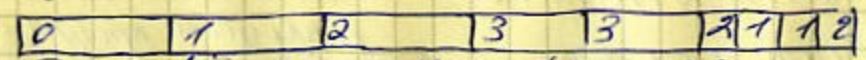
б) DYNAMIC [, m]



вначале всё так же, а потом новую порцию дают тому, кто быстрее освободился

В конце - один работает, остальные простаивают

в) guided [, m]



блоки всё меньшего размера

2) runtime

меняет способ в динамике или выбрал свой

3. Секции

Нет, когда нужно описать конечный параллелизм, нет в программе

```
!$ OMP SECTIONS
!$ OMP SECTION
      <1>
!$ OMP SECTION
      <2>
!$ OMP SECTION
      <3>
! OMP END SECTIONS
```

(NOWAIT) - нет барьеров

4. Single

Нет. фрагмент кода должен быть исполнен 1 раз (неважно, кем)

```
!$ OMP SINGLE
      <...>
!$ OMP END SINGLE
```

Классы переменных

- общие (Shared) A
- локальные (private) B


```
x = get_OMP_thread_num();
```

point x

x - private => 0, ..., N-1

x - shared => N чисел в том же диапазоне

Директива синхронизации

!$OMP BARRIER - ждать всех

!$OMP CRITICAL <имя>
<...>
!$OMP END CRITICAL <имя>

- много крит. секций → много потер. участков → ↓ производ.
- много лок. перем. → трата памяти

80% времени → фрагмент переводим в OMP

10% — OMP

openmp.org

MPI (Message Passing Interface) 17.11

Номер процесса
0, ..., n-1

SPMD (single program - multiple data)

MPI_

```
#include <mpi.h>
```

- номер пр-опир
- — " — -колич.
- идентификатор сообщения (тег, 0, 32767)

тег нужен программисту, можно всем давать одинаковый (номер, 0)

int...
MPI_SUCCESS - всё хорошо

```
int MPI_Init(int *argc, char **argv)
{...
int MPI_Finalize(void)          //всё д. быть между
```

MPI_Comm_rank (MPI_Comm comm, int *rank)

Группа процессов - перечисление входящих в группу процессов

Коммуникатор помогает процессам общаться
MPI_COMM_WORLD - главный коммуник., объединяет все процессы приложения

Внутри 1 группы (комм-ра) процессы имеют собственные номера

rank - номер процесса

MPI_Comm_size(MPI_Comm comm, int *size)

Далее можно распараллеливать:

if(rank == 0){...}

double MPI_Wtime(void) – время, прошедшее с нек. момента

Точка – точка

MPI_Send(void *buf, int count, MPI_Datatype dt, int dest, int msgtag, MPI_Comm comm)

Оба процесса должны принадлежать одному коммуникатору

Блокирующая

int → MPI_INT
double → MPI_DOUBLE

MPI_Bsend
buffer,

буферир. посылка: нужно завести спец. массив данных, и при пересылке сообщ. будет скопировано в буфер – нет блокировки

MPI_Ssend
Синхронная – ждём, пока сообщ. не примется

MPI_Rsend
Ready – получ-ль должен гарантировать, что принимающий процесс готов, и сообщ. без проверки посылается прин. процессу

MPI_Recv(void *buff, int count, MPI_Datatype dt, int source, int msgtag, MPI_Comm comm, MPI_Status *status)

Эл-нт MPI_Status – структура, содерж. минимум 3 поля: status.MPI_SOURCE /N отпр
status.MPI_TAG /тэг сообщ.
Status.MPI_ERROR /код ош.

Принцип. процесс не обязан знать отправителя и тэст → можно указать в аргументах MPI_ANY_SOURCE и MPI_ANY_TAG

блокирующая ! пока не получится сообщ.

MPI_Get_count(MPI_Status *status, MPI_Datatype dt, int *count) – сколько данных было передано

MPI_Send MPI_Send
MPI_Recv MPI_Recv

MPI_Isend (..., MPI_Request *req) – для проверки завершённости
:
:
Ibsend
Issend
Irsend

MPI_Irecv (...)

MPI_Wait (MPI_Request *req, MPI_Status *status) – ожидание завершения неблокир. опер.

Неблокир. – нужен для deadlock

Waitall
Waitany
Waitsome

MPI_Test (req, int *flag, status)
проверка, завершилась ли операция req

MPI_Testall
Testany
Testsome

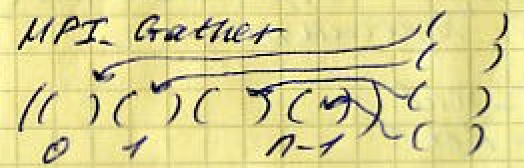
MPI_Sendrecv(...)
deadlock завершается аппаратно

Коллективные взаимодействия -

участвуют все процессы коммуникатора

MPI_Bcast (void *buf, int count,
MPI_Datatype dt, int source,
MPI_Comm comm)
рассылает всем процессам в comm

MPI_Gather



MPI_Scatter — обратная к Gather
(все посылают одному) - один посыл. всем разные ч. массива

MPI_Reduce (void *sbuf, void *rbuf,
int count, MPI_Datatype dt,
MPI_Op op, int root, MPI_Comm comm)

sbuf
(00...)
(01...) op

MPI_Barrier(MPI_Comm comm)
барьерная синхронизация

MPI_Comm_split(MPI_Comm comm,
int color, int key, MPI_Comm *newcomm)

Новые коммуникаторы:

rank % 2 - с чётным номером - один комм, с нечётным - другой

rank < $\frac{n}{2}$

MPI_Comm_free(MPI_Comm comm)
можно удалить ненужный коммуник.

MPI 2

1) динамич. порождение процессов
2) односторонние коммуникации
send-receive
put/get 1 активный участник

put → get

3) параллельный ввод/вывод
4) расширение ф-ий коллект. операций
5) C++

1.12 Введение в теорию анализа структуры парал. прогр-я

Пример перемножение матриц

```
for(i=0; i<n; ++i)
  for(j=0; j<n; ++j)
    for(k=0; k<n; ++k)
      A[i][j] = A[i][j] + B[i][k] + C[k][j];
```

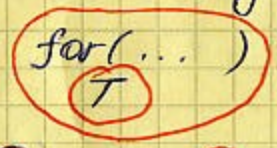
порядок циклов (i,j,k) ⟷ (k,j,i)

Для данной прогр. ∀ из 6 порядков допуст. процув. может в ~5-7 раз отличаться в завис. от порядка циклов

Вариант (k,i,j) и (i,k,j) похожи лучше всего, этот цикл по j был внутренним

Графовые модели

Вершины: циклы, операторы, лин. участки кода, тело цикла, отдельные срабатывания оп-в



(отд. срав. оп-ров)

n (# операций цикла)

Дуги: – операц. отношения (по передаче управления)

A → B ⇔ B м.б. выполнено сразу после A

– информационное (по передаче данных)

4 основные графовые модели:

- Граф управления программы

(операторы; операц. отношения)

```
for(i=0; i<n; ++i)
{   A[i] = A[i-1]*p + 17;
    B[i] = A[i] - t;
}
```

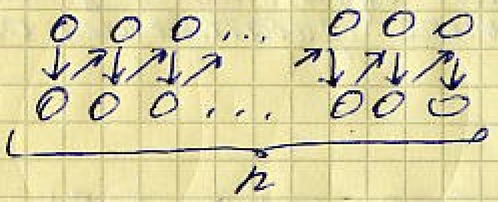
Граф управления

- Информационный граф пр-ма

(операторы; инф. отн-я)

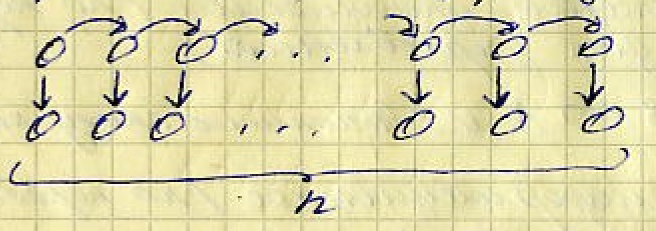
Операционная история

(срабатывание опер-в, операц. отношения)

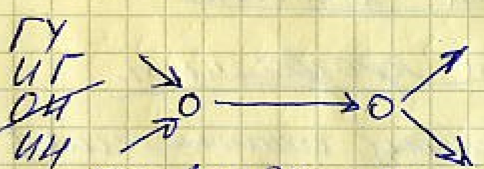


Информационная история

(сраб. опер-в, информац. отн-я)

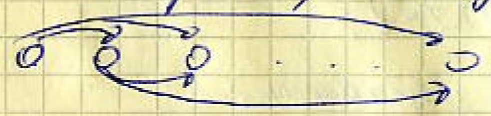


ИИ может быть без дуг: $a_i = b_i + c_i$



ОИ всегда линейной

ИИ: 20 вершин, 200 дуг



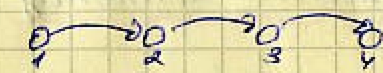
$\frac{n(n-1)}{2} < 200$

=) не может быть

ГУ и ИГ строятся по посл-ти опер-в

ИИ и ОИ - по динамике

Операц. отношения



$x = a + 12;$
$y = x + t - 8;$
$t_1 = y + 2;$
$t_2 = y + a - 8;$

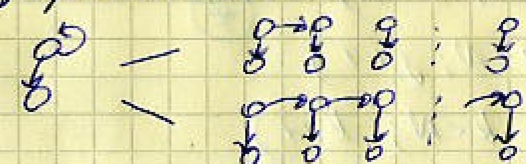
информ. независимость
ресурс параллелизма

поменять порядок не можем

Информ. зав-ть - критерий эквив-ти преобр.

Критерии:

- компактность (+ИГ)
- информативность (+ИИ)



- реализуемость (+ИГ)

Линейный класс

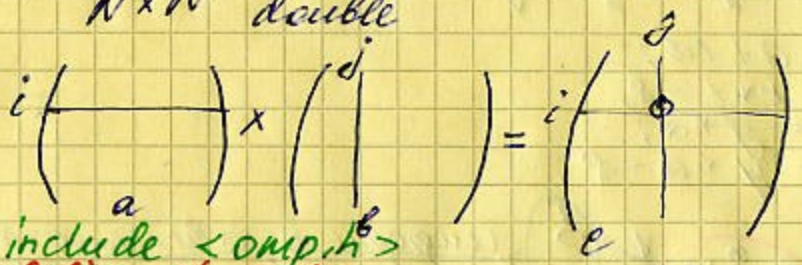
Граф алгоритма - параметрич. заданная инф. история

+ компактность
+ строится по тексту пр-ты (реализуемость)

8.12

Задача перемножения матриц

$N \times N$ double




```
#include <omp.h>
#define N 4096
for(i=0; i<N; i++)
    for(j=0; j<N; j++)
    {
        a[i][j] = (i+j)*0.0001;
        b[i][j] = (i-j)*0.0001;
        c[i][j] = 0.0;
    }
t1=omp_get_wtime();
#pragma omp parallel shared(a,b,c) private(i,j,k,size)
{ for(i=0; i<N; i++)
    for(j=0; j<N; j++)
      for(k=0; k<N; k++)
        c[i][j] += a[i][k]*b[k][j];
}
```

(*)

- ресурс параллелизма
- распределить операции
- выбрать технологию
- распред. данных (MPI)
- пересылка данных (MPI)
- реализация алгоритма
- отладка
- эффективность и масштабирование

(*)

```
size = omp_get_num_threads();
#pragma omp for schedule(static)
  // разбиваются на равные части
t2 = omp_get_wtime();
for(i=0; s=0.0; i<N; i++)
  for(j=0; j<N; j++)
    s += c[i][j]
printf("N=%d, Nproc=%d, sum=%lf,
  time=%lf\n", N, size, s, t2-t1);
```

СКИФ МГУ „Чебышёв"

2×4 ядра = 8 нитей

Коммилятор Intel 11.1

-O3 -openmp

# нитей	1	2	4	8	16
время	164	89.8	59.5	36.9	53.4

Распределение данных

- строчная / столбцовая

строчное

j-й столбец в надо передать процессу, вычисляющему C_{ij}

Заведем матрицу-буфер $n \times 1$, buf( ) куда и будем пересылать столбец в блочный столбец надо рассылать всем процессам: MPI_Bcast

```
#include <mpi.h>
MPI_Init (&argc, &argv);
MPI_Comm_rank (MPI_COMM_WORLD, &rank)
MPI_Comm_size(MPI_COMM_WORLD, &size);
myrows = mycols = rows = cols = (N-1)/size +1;
last_rows = last_cols = N - rows *(size - 1);
if (rank == size - 1)   // посл. процесс
  myrows = mycols = last_rows;
double **a = (double **) malloc (sizeof (double)*
  myrows );
 for (i=0; i<myrows; i++)
     a[i] = (double *) malloc (sizeof (double)*N);
 b.. c..
double *buf = (double *) malloc (sizeof (double)
                                 * cols + N);
```

```
/* инициализация */
MPI_Barrier ( MPI_Comm_WORLD );
t1 = MPI_Wtime();
 for (m=0; m<size; m++)
   if (rank == m)
     for (j=0; j< mycols; j++)
       for (k=0; k<N; k++)
           buf [k+j*N] = b[k][j];
 if (m == size-1) l = last_cols*N;
 else  l = cols * N;
 MPI_Bcast (buf, l, MPI_Double, m,
                    MPI_Comm_WORLD);
 if (m == size-1)
     jmax = last_cols;
 else jmax = cols;
  for (i=0; i<myrows; i++)
    for (j=0; j<jmax; j++)
      for (k=0; k<N; k++)
          c[i][j] += a[i][k] * buf [k+j*N];
}
MPI_Barrier( MPI_Comm_WORLD)
t2 = ...
```

```
for (i=0; sum=0.0; i<myrows; i++)
  for (j=0; j<N; j++)
    sum+=c[i][j];
MPI_Reduce(&sum, &s, 1, MPI_DOUBLE,
           MPI_SUM, 0, MPI_COMM_WORLD);
if (rank==0) printf(...
```

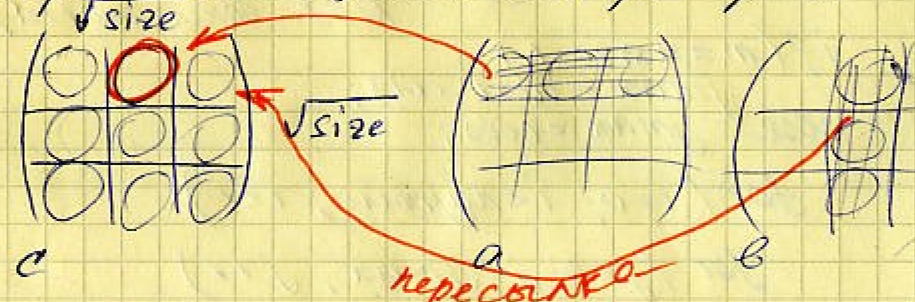
#thr	1	2	4	8	16	32	64	128
time	155.3	88.2	79.2	63.2	7.5	2.6	2.34	1.60
	164.1		17,36	10.2 (9#)	8.27	2.3 (#25)	0.98	0.67 (#121)

#thr	256	512	1024
time	3.77	5.6	12.4
	0.49	0.64 (#484)	0.93

(затраты на пересылку данных)

данные для 2 алгоритма (эффект. если # = n^2)

- блочно-столбцовый тип распр-я



Опер. $N^2(2N-1)$

Пересылки size $cols*N = \frac{N^2}{size} * size = N^2$

$N^2(size-1)$ общий объем пересылки для 1 алгоритма

$\left(\frac{N}{\sqrt{size}}\right)^2 = \frac{N^2}{size}$ (размер пересыл. блока)

$2(\sqrt{size}-1) * \frac{N^2}{size} * size = 2(\sqrt{size}-1)*N^2$ для 2 алгоритма

пересылок:

size + (size-1) в 1 алгоритме

$2(\sqrt{size}-1)*size$ во 2 алгоритме

Объем буфера в 1 алг. $\frac{N^2}{size}$

во 2 алг. $2*\sqrt{size}\frac{N^2}{size}$

в $2*\sqrt{size}$ раз больше объём памяти