Динамическая балансировка загрузки процессоров

Якобовский Михаил Владимирович
проф., д.ф.-м.н.
Институт прикладной математики
им. М.В.Келдыша РАН, Москва

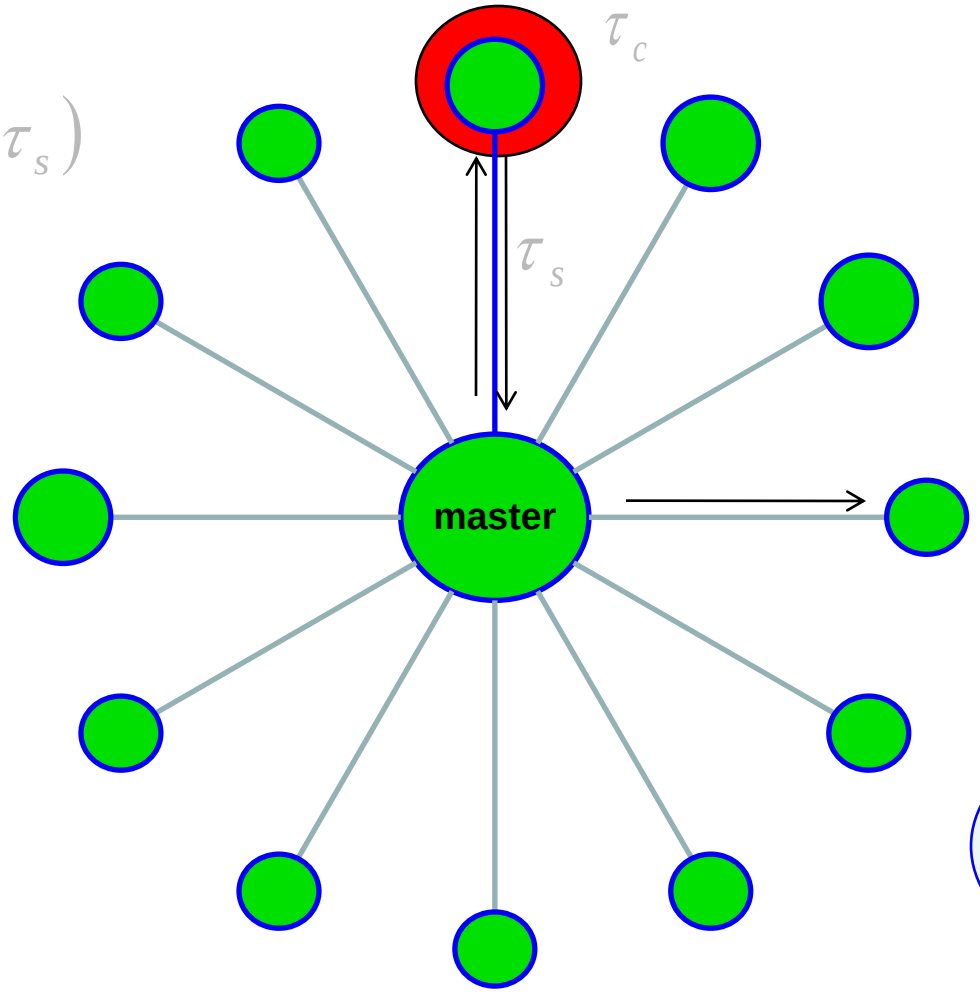
Метод коллективного решения

Решение набора независимых друг от друга задач:

- Табулирование функций
- Решение задач методами Монте-Карло
- Численное интегрирование гладких многомерных функций
- …

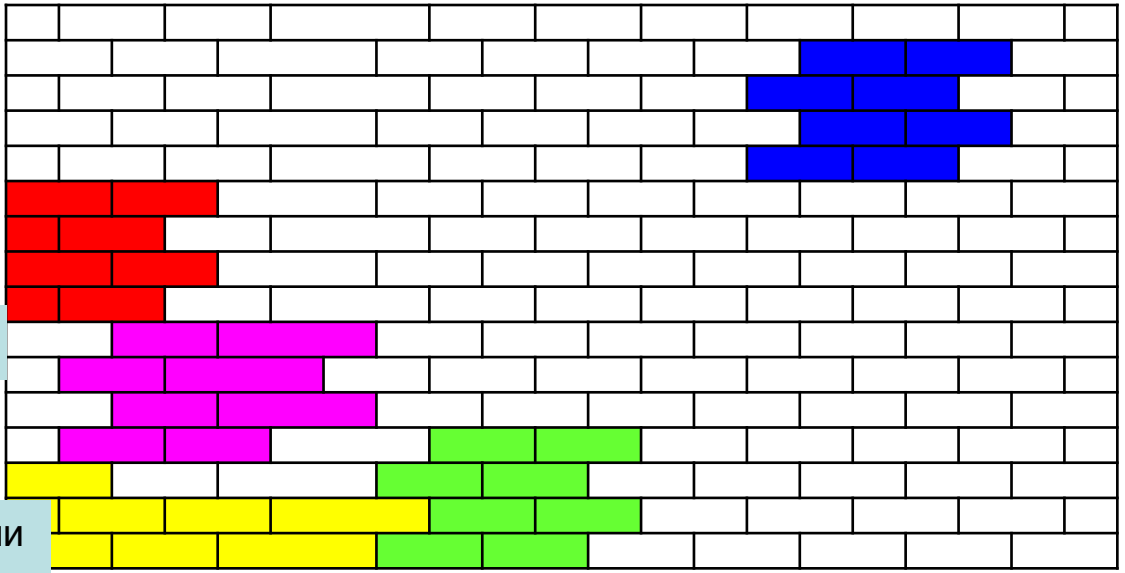
Метод коллективного решения

$$T_p = \frac{N}{p}(\tau_c + \tau_s)$$



$$p_{\max} = \frac{\tau_c}{\tau_s}$$

Пример модельной задачи: Укладка паркета



$$T_1(kn) = \tau_c kn$$

$$T_p(kn) = \frac{kn}{rp}\left(r\tau_c + \tau_s\right)$$

Число порций

Обработка порции

Обмен данными

$$S_{p=\frac{r\tau_c}{\tau_s}}(kn) = \left(\frac{r\tau_c}{\tau_s}\right)^2 \frac{1}{1+\frac{r\tau_c}{\tau_s}} = p_{\max}\frac{1}{1+\frac{1}{p_{\max}}}$$

$$E_{p=\frac{r\tau_c}{\tau_s}}(kn) = \frac{1}{1+\frac{\tau_s}{r\tau_c}}$$

$$p_{\max} = \frac{r\tau_c}{\tau_s}$$

r – размер порции

Как правильно?

r – размер порции

$$p_{\max} = \frac{r\tau_c}{\tau_s} \quad \textit{или} \quad p_{\max} = \frac{r\tau_c}{r\tau_s} = \frac{\tau_c}{\tau_s}$$

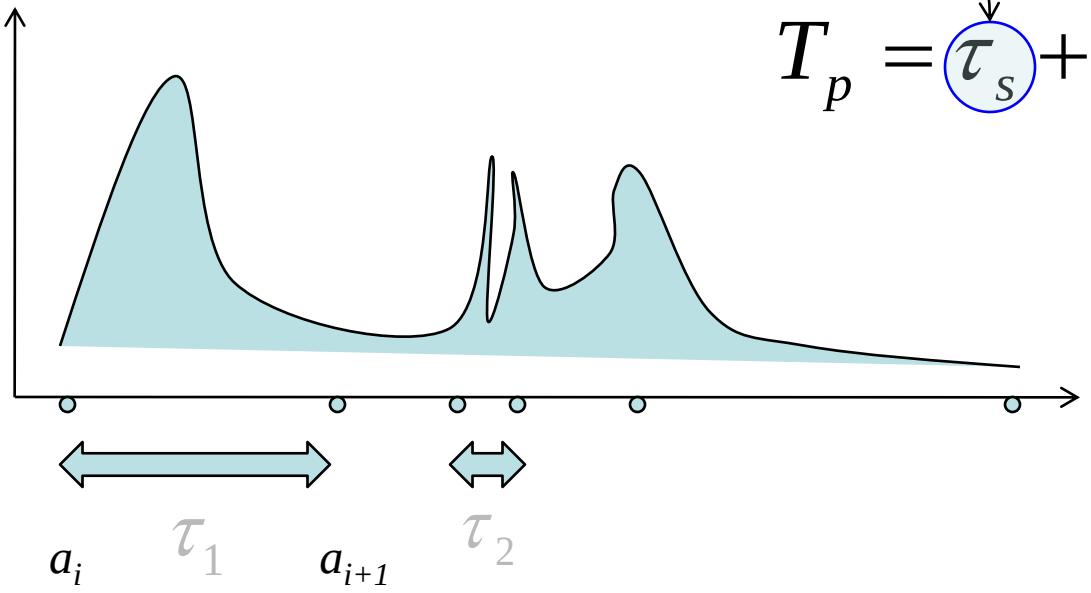
Зависит ли время передачи данных от размера порции (задания)?

Вычисление определенного интеграла

Send(a_i); Send(a_{i+1}); Recv(s);

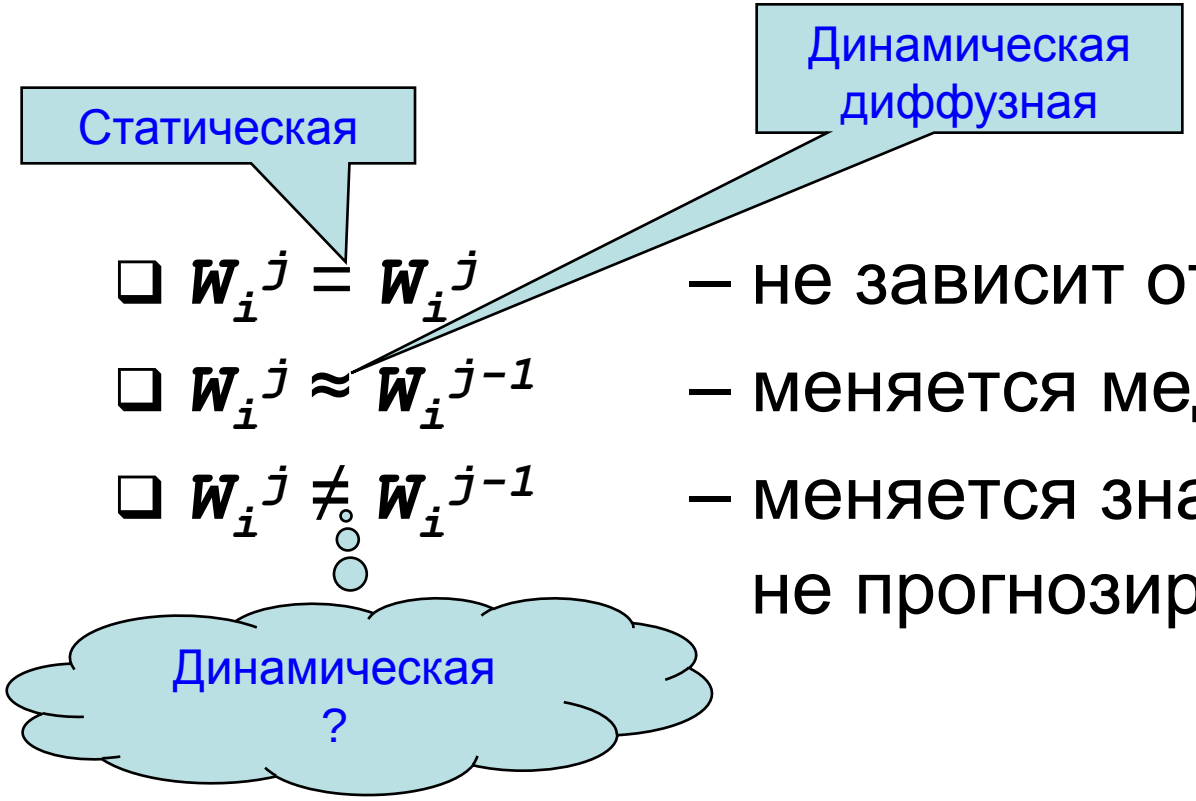
$$I = \int_A^B f(x)dx = \sum_i \int_{a_i}^{a_{i+1}} f(x)dx$$

$$T_p = \tau_s + \max_i \tau_i$$



Стратегии балансировки загрузки

W_i^j - вычислительная нагрузка, ассоциированная с узлом сетки i на шаге j



- $W_i^j = W_i^j$ – не зависит от времени
- $W_i^j \approx W_i^{j-1}$ – меняется медленно
- $W_i^j \neq W_i^{j-1}$ – меняется значительно и не прогнозируемо

Причины возникновения дисбаланса

- неудачное начальное распределение элементарных заданий
- изменение эффективной производительности процессоров
- изменение трудоемкости обработки элементарных заданий (например, вычислений в узлах сетки) уже в ходе выполнения работы

Свойства элементарных заданий

Статические

- Не всегда можно заранее точно предсказать трудоемкость обработки узлов сетки

Динамические

- Трудоемкость обработки каждого из узлов расчетной сетки может отличаться на разных моментах модельного времени

Свойства процессоров

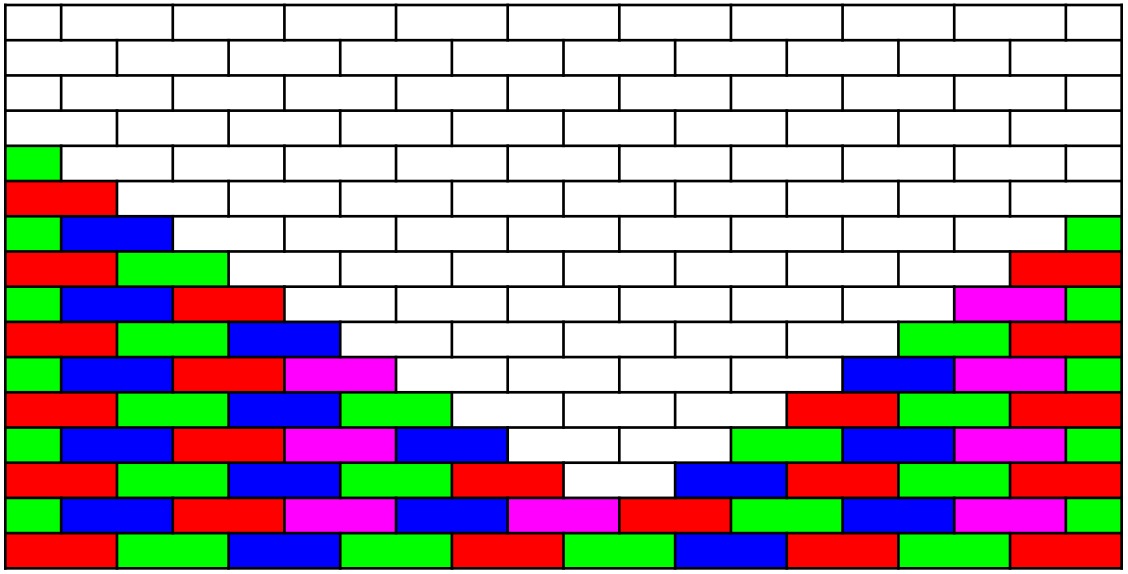
Статические

- Процессоры могут обладать разной, неизвестной заранее производительностью

Динамические

- Эффективная производительность процессоров может меняться с течением времени

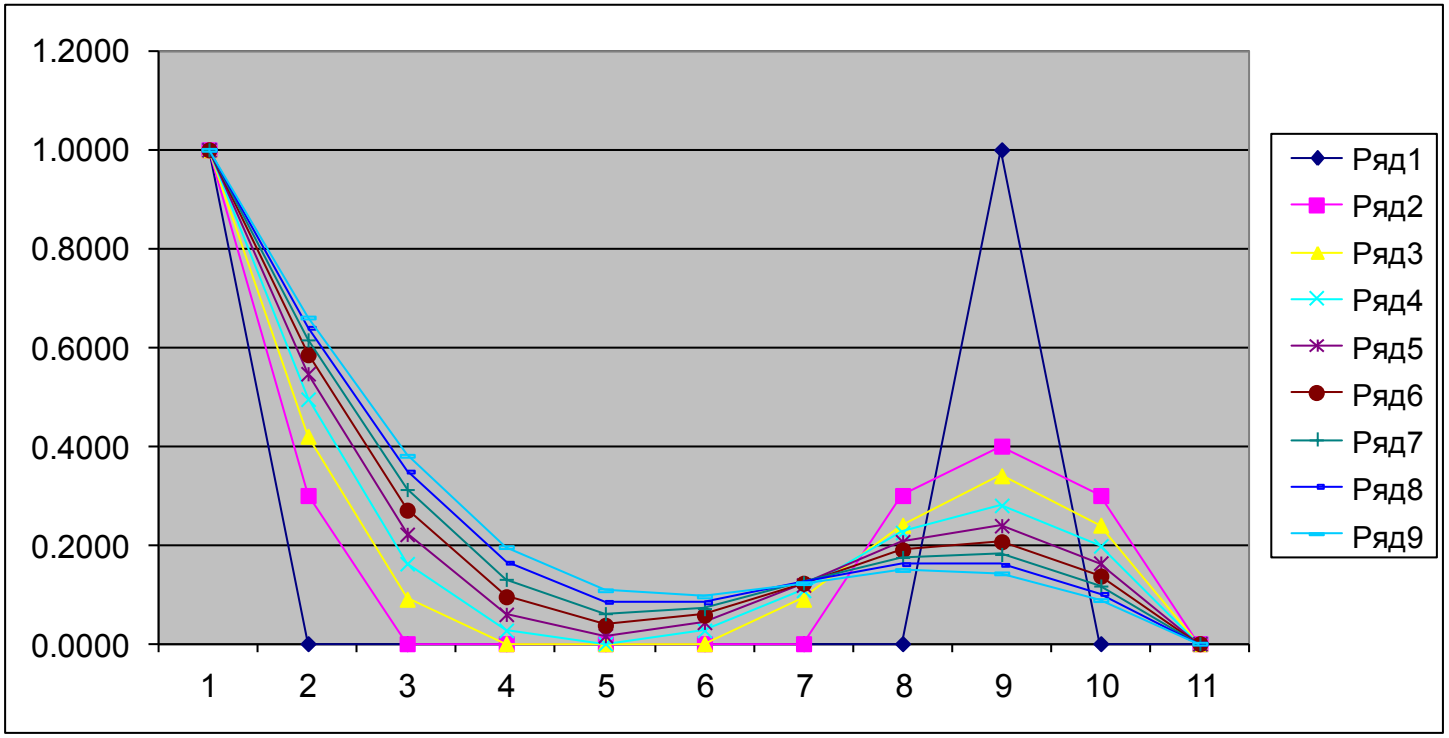
Стена Фокса



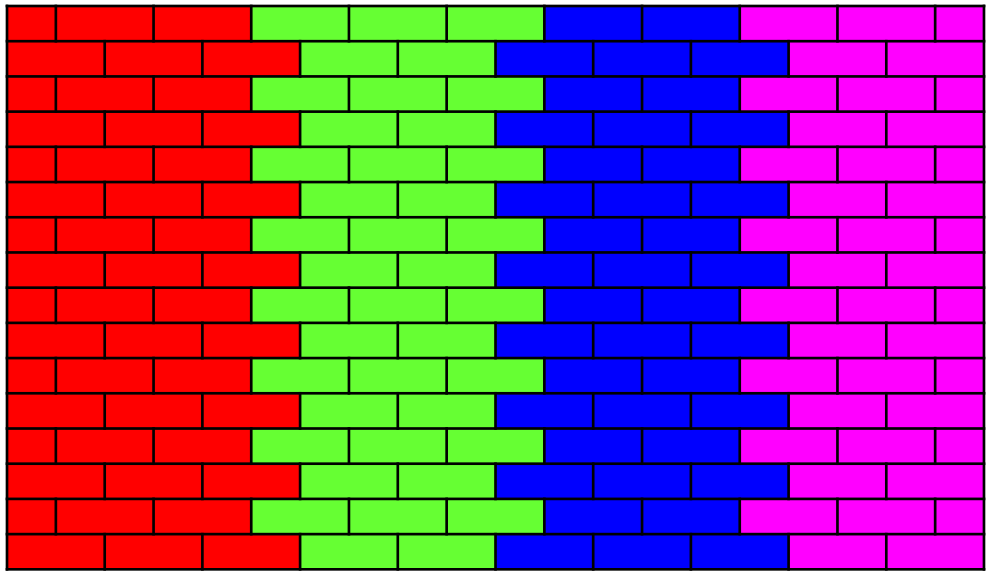
n – ширина стены

κ – высота стены

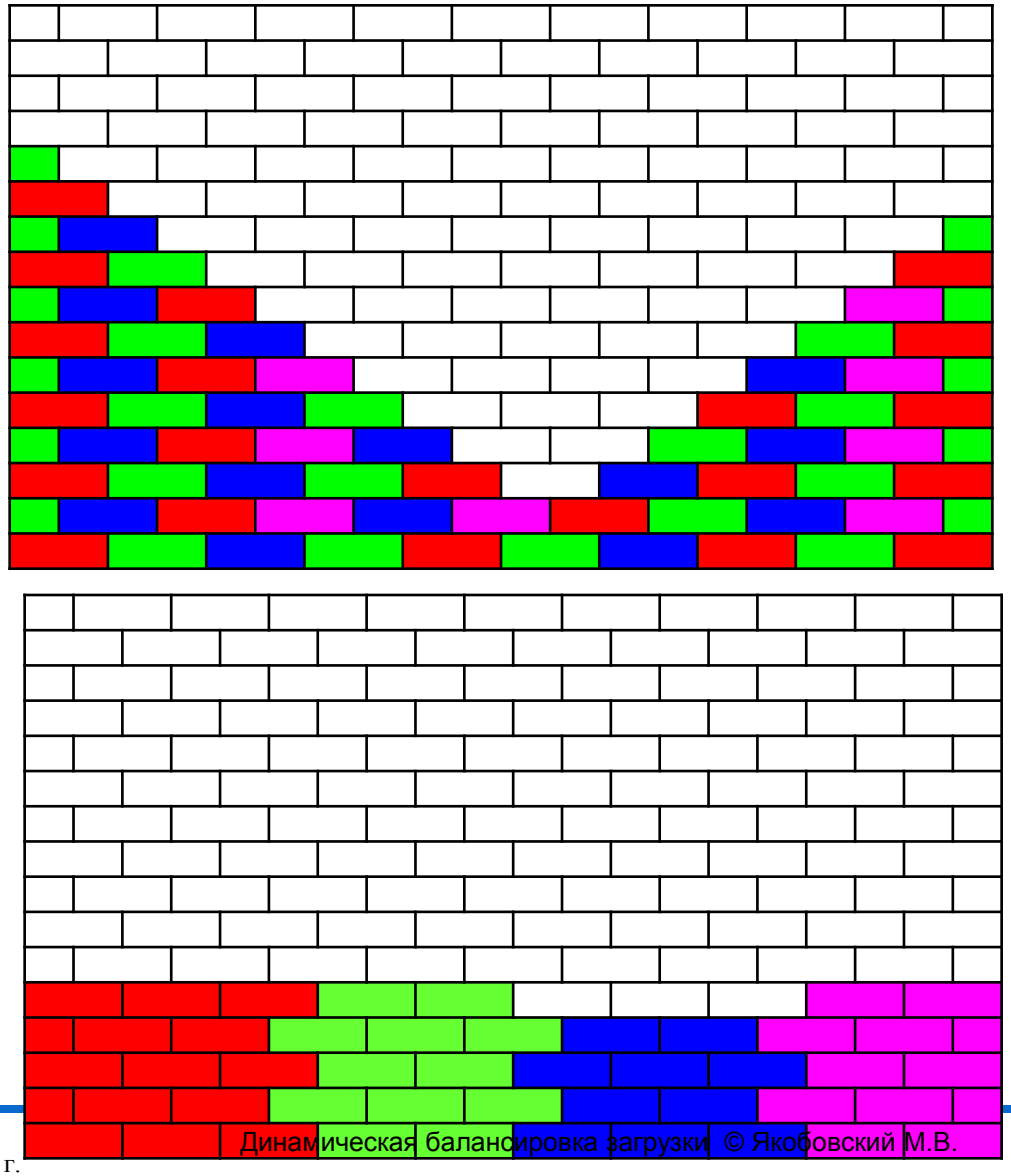
Рраспространение тепла по стержню



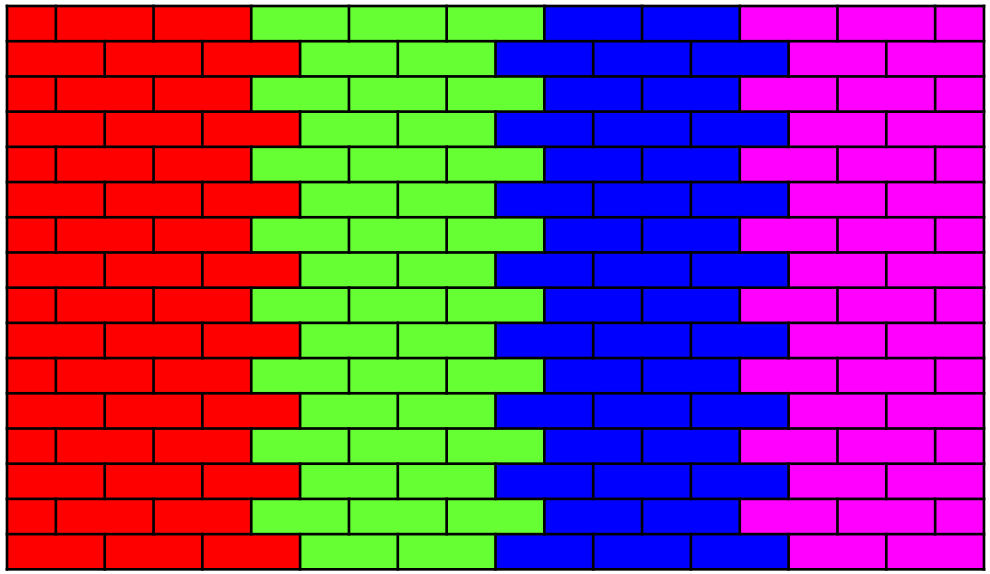
Метод геометрического параллелизма



Метод геометрического параллелизма



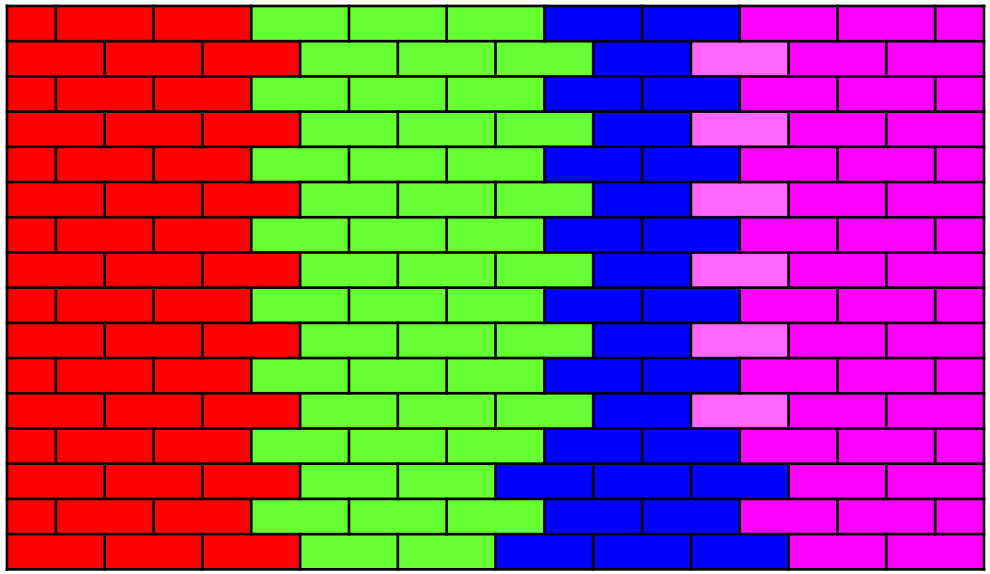
Метод геометрического параллелизма



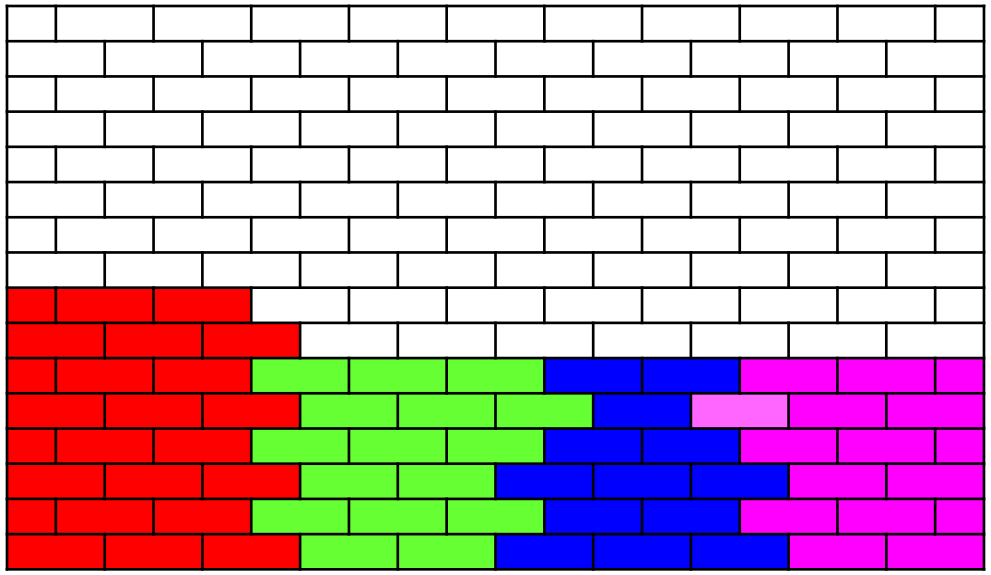
$$T_1(kn) = \tau_c kn \qquad T_p(kn) = \tau_c \frac{kn}{p} + 4k\tau_s$$

$$S_p(kn) = p \frac{1}{1 + 4\frac{p}{n}\frac{\tau_s}{\tau_c}} \qquad E_p(kn) = \frac{1}{1 + 4\frac{p}{n}\frac{\tau_s}{\tau_c}}$$

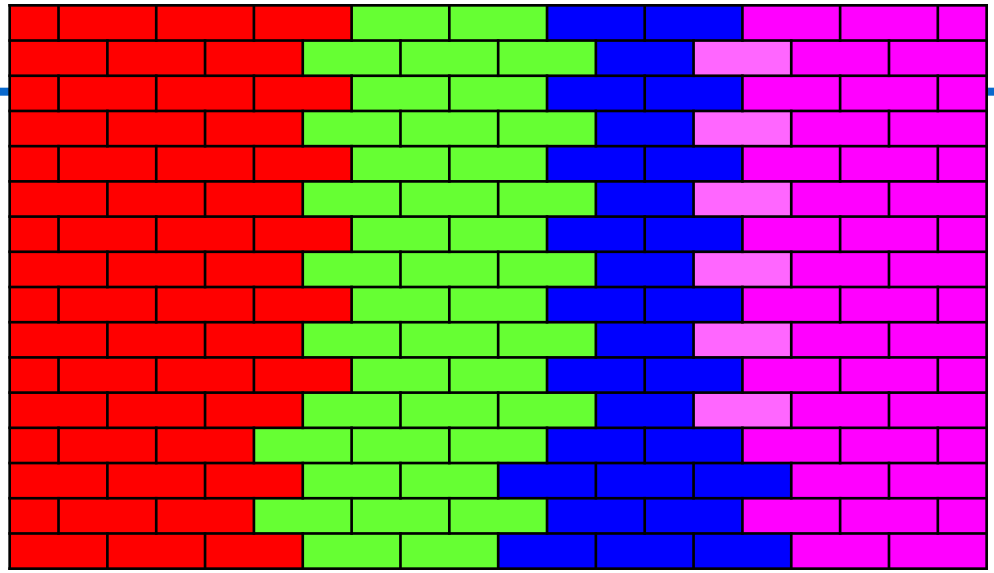
Диффузная балансировка загрузки

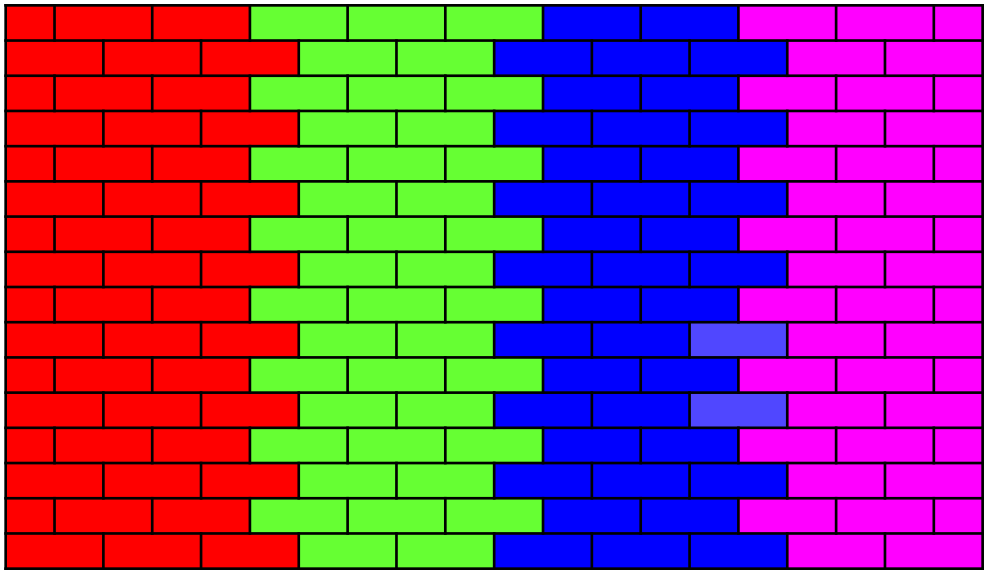


Диффузная балансировка загрузки



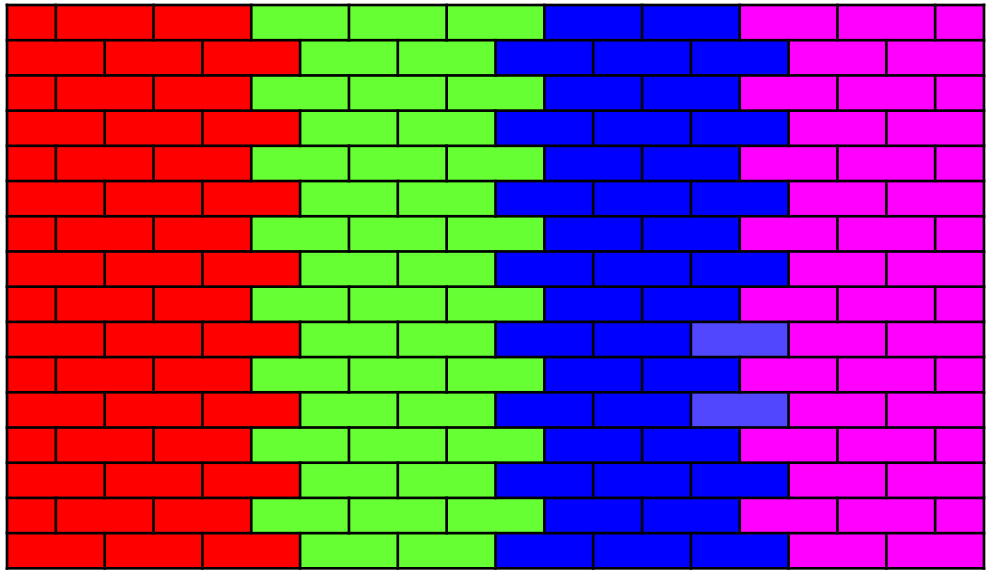
Диффузная балансировка загрузки





Статическое распределение

Статическое распределение



если Дисбаланс вычислительной нагрузки обусловлен разными производительностями процессоров

$$n_i' = n \frac{\frac{n_i}{t_i}}{\sum_{j=1}^{p} \frac{n_j}{t_j}}$$

если Дисбаланс обусловлен различной трудоемкостью обработки узлов сетки

процессоры обладают одинаковой производительностью

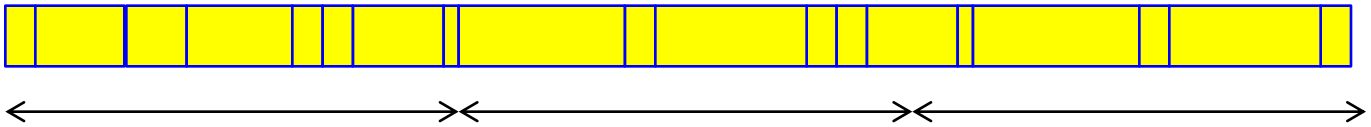
Общая вычислительная нагрузка определяется как

$$T = \sum_{k=1}^{p} t_i$$

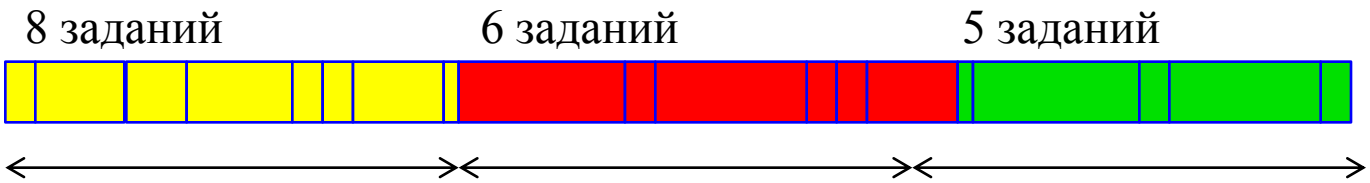
$$T_{mid} = \frac{T}{p}$$

Алгоритм распределения заданий по одинаковым процессорам

```
m=0;    t=0;
for(j=0;j<n;j++)
{
   t=t+σ_j
   if(t ≥ T_mid)    {        n_i''=j-m;    m=j;    t=0;    }
}
```



Каждому процессору назначается множество очередных заданий, суммарная трудоёмкость которых равна средней по всем процессорам



Перераспределение узлов сетки

номер процессора i	1	2	3	4
n_i	5	10	14	7
$t_{i,}$	1	2	1	3
$n_{i,,}$	7	7	19	3
n_i	10	10	11	5

Два разных решения

Перераспределение узлов сетки

номер процессора i	1	2	3	4
n_i	5	10	14	7
t_i	1	2	1	3
n_i'	7	7	19	3
n_i''	10	10	11	5

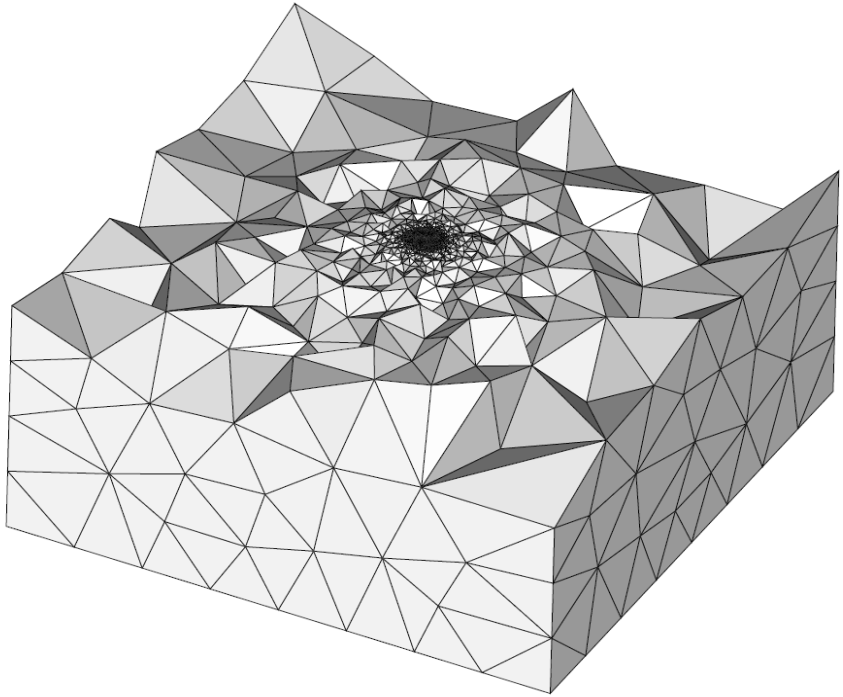
Два разных решения
Какое верно?

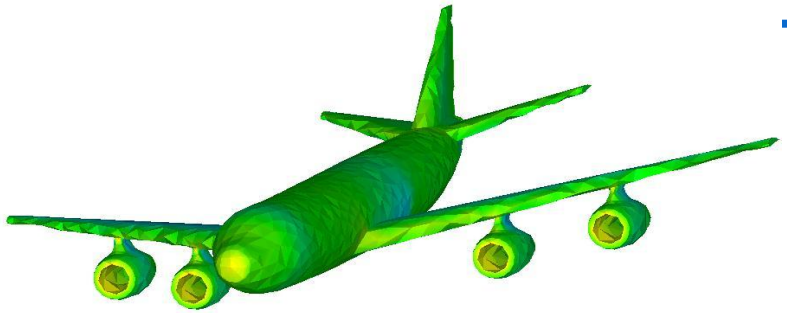
Перераспределение узлов сетки

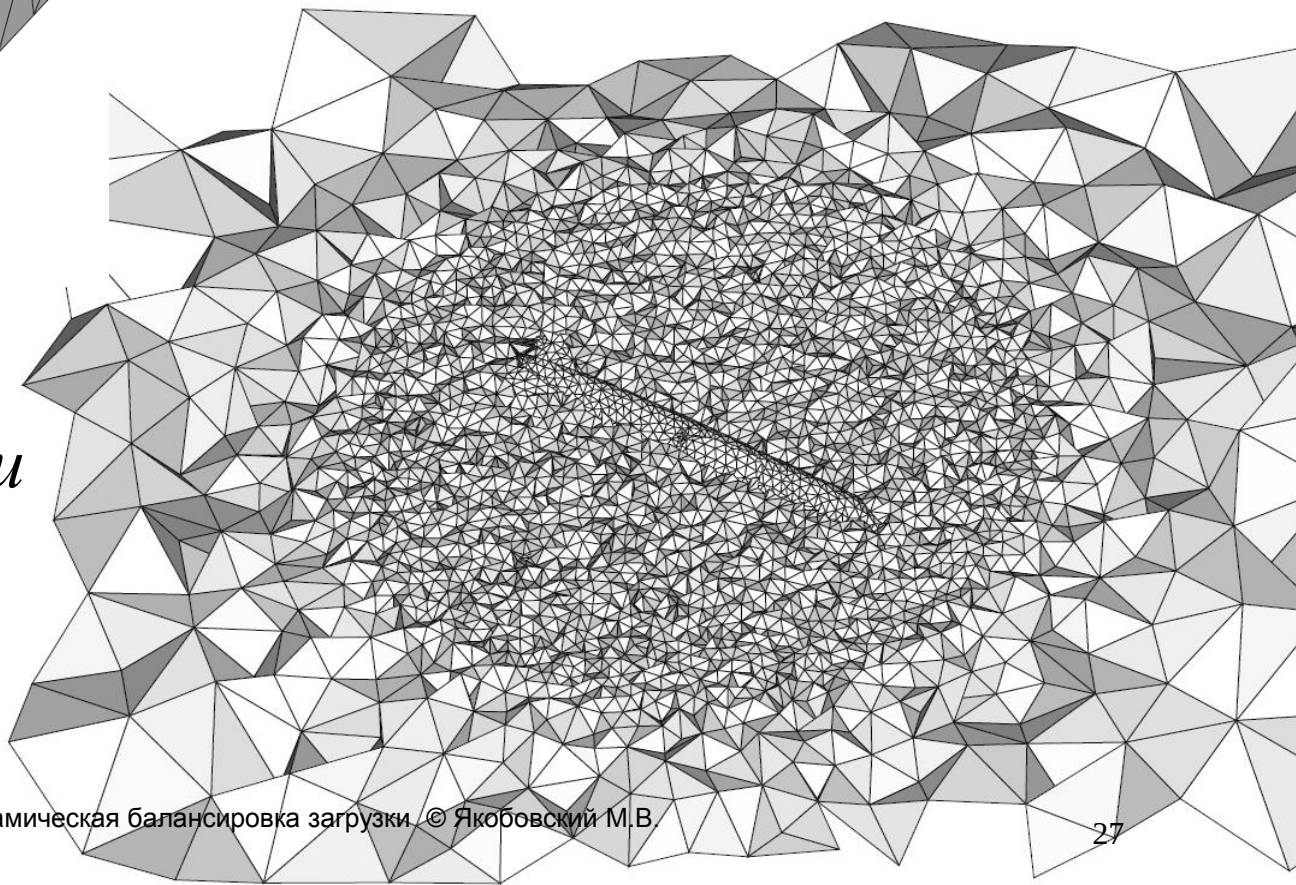
номер процессора i	1	2	3	4
n_i	5	10	14	7
t_i	1	2	1	3
n_{i_1}	7	7	19	3
n_i	10	10	11	5

Два разных решения
Какое верно?
Почему одно из них верно?

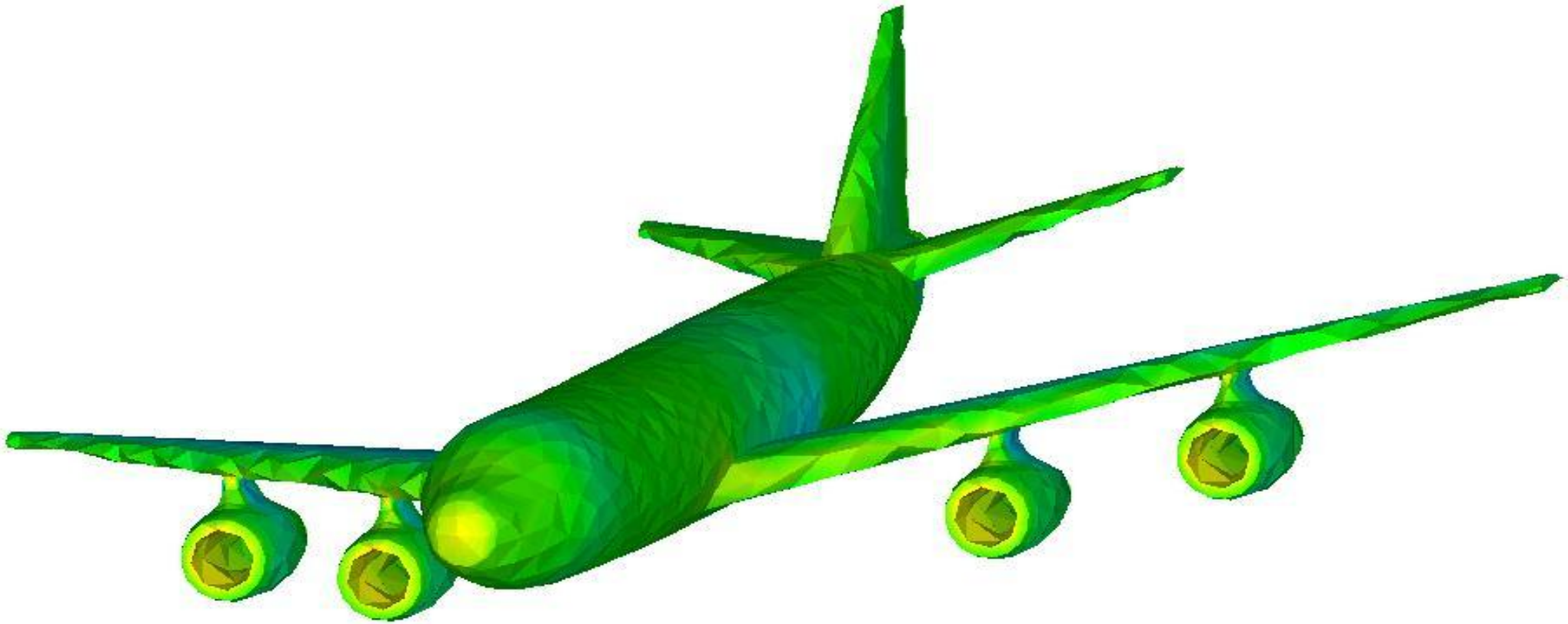
Сетки Неструктурированные





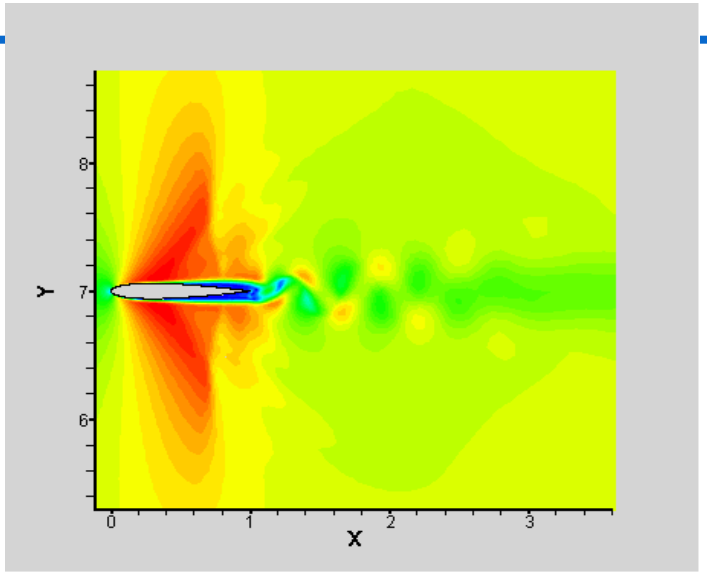


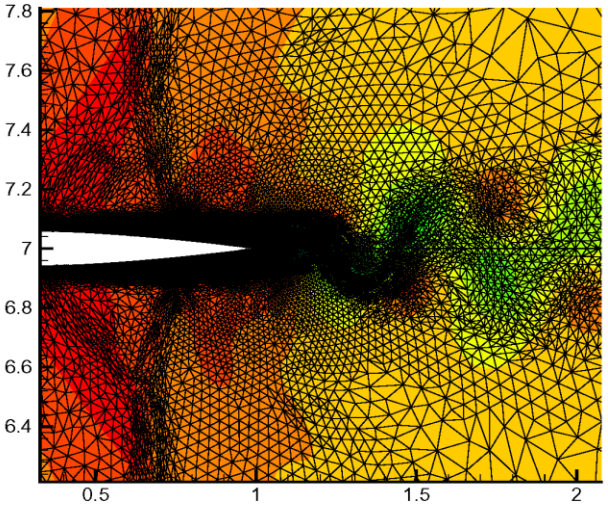
Тетраэдральные сетки
10^8 узлов



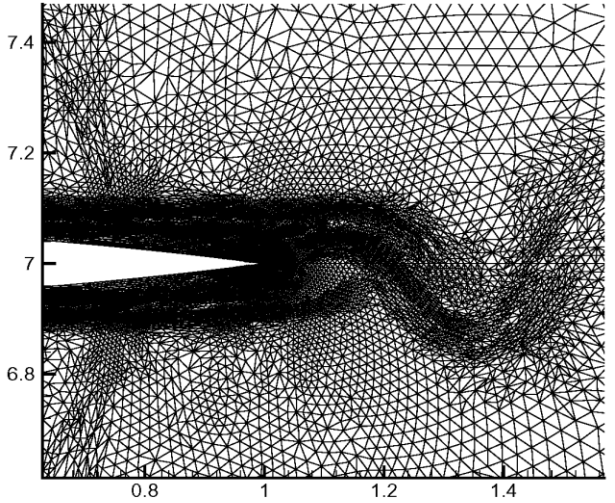
Использование адаптивной сетки

Обтекание профиля NACA0012
(M=0.85, Re=10^4)
под нулевым углом атаки:





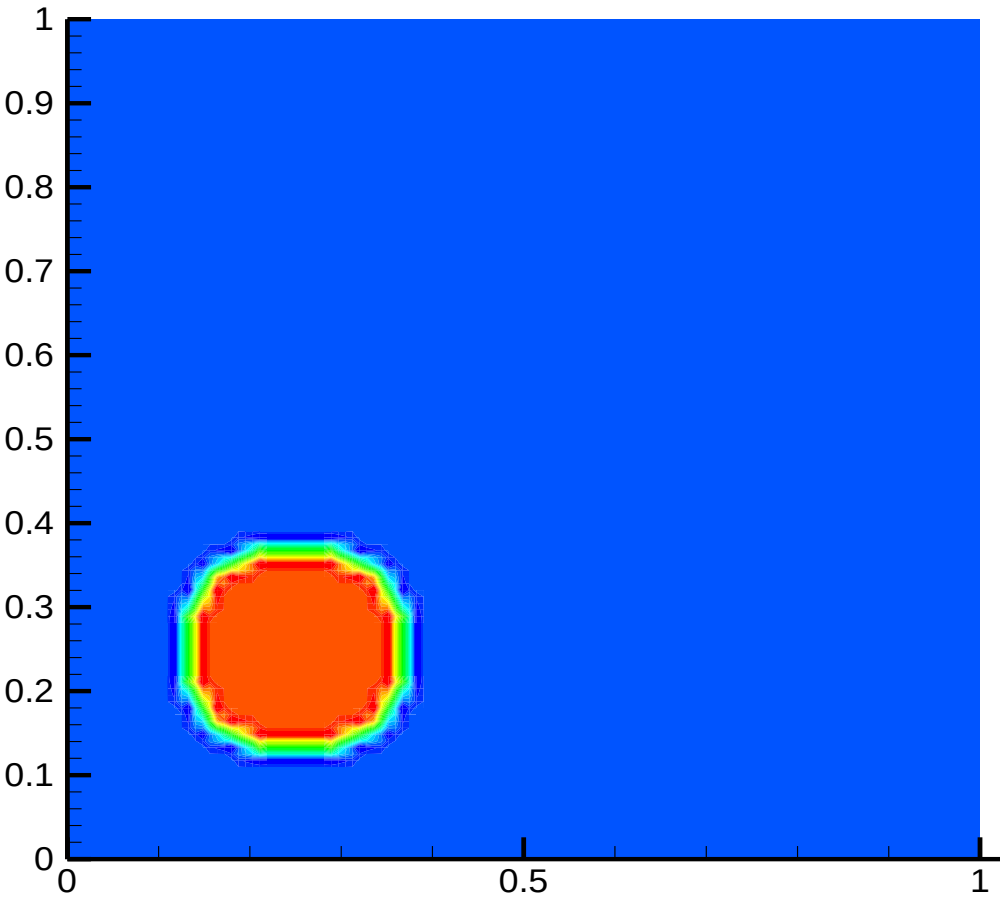
Поле продольной скорости

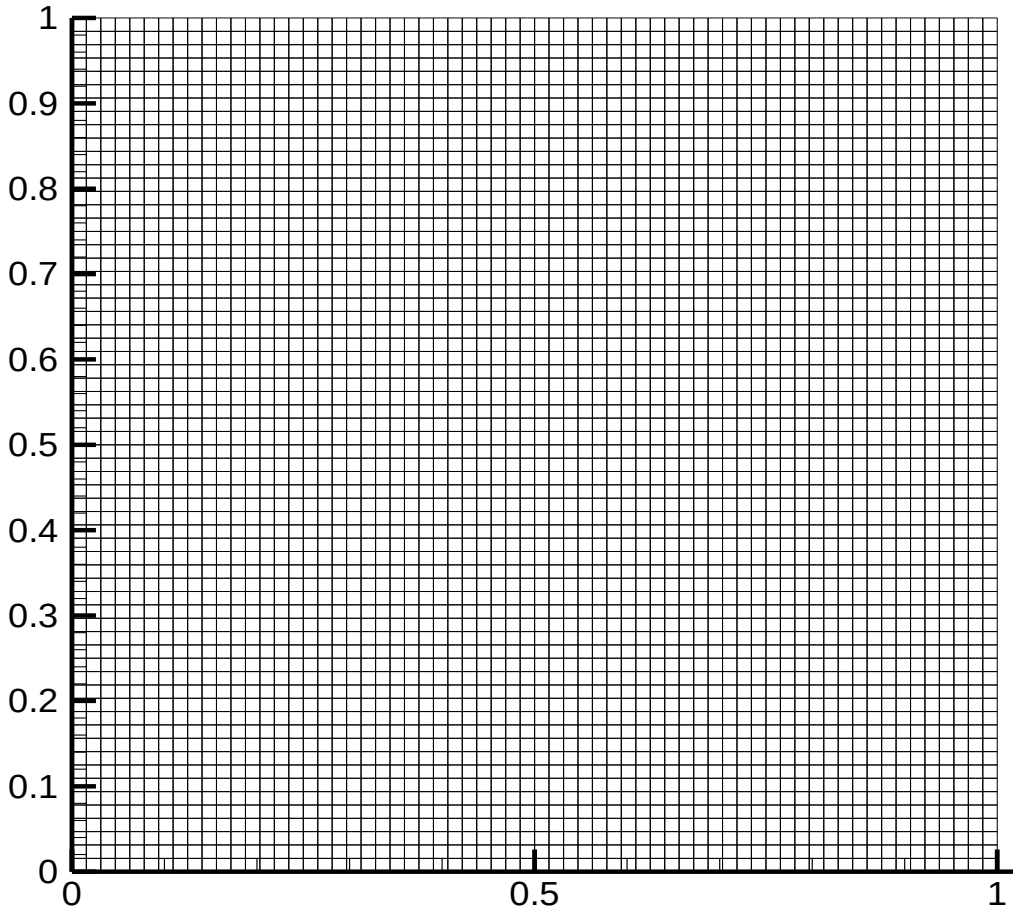


Фрагмент сетки

Равномерная сетка

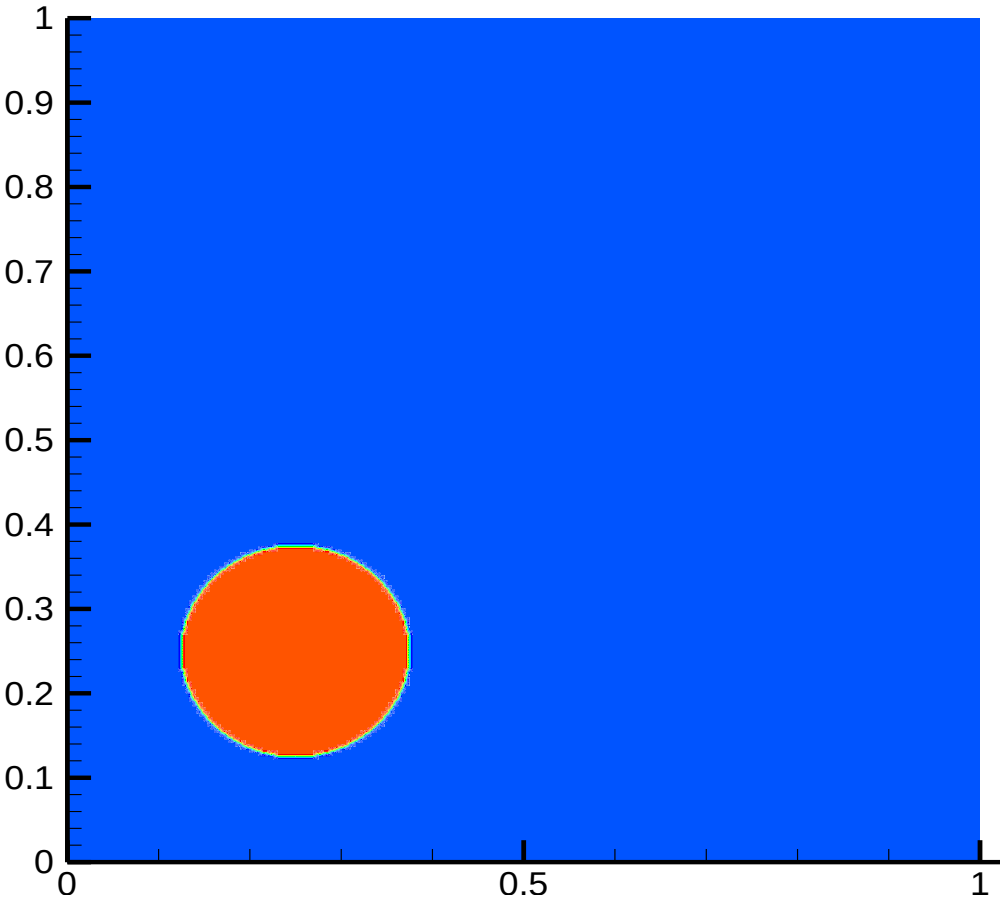
Слева – ??*круглое*?? пятно примеси

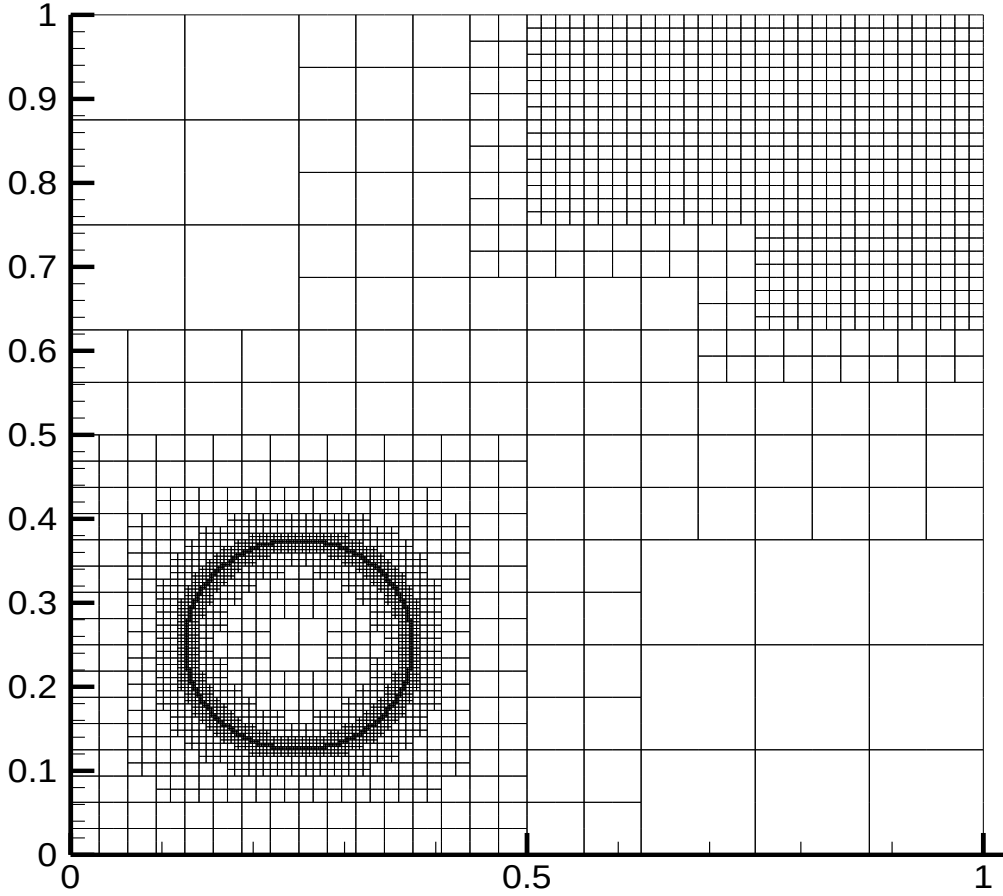




Адаптивная сетка

Слева – ***круглое*** пятно примеси

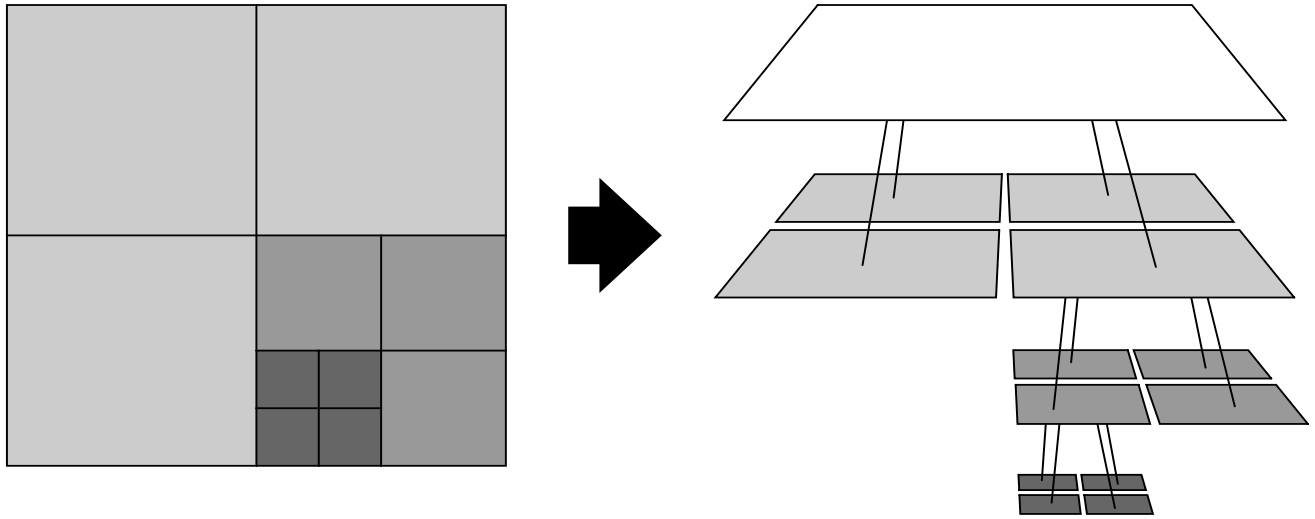




Адаптивные декартовы сетки

- Вначале сетка состоит из одной прямоугольной ячейки
- Каждая ячейка может быть разделена на четыре ячейки одинакового размера
- Если ячейки когда-то составляли одну ячейку, то они могут быть объединены обратно
- Каждая ячейка хранит величину, описывающую среднее значение неизвестной функции в пределах ячейки (метод конечных объёмов)

При данных предположениях сетку удобно хранить в виде четверичного дерева:

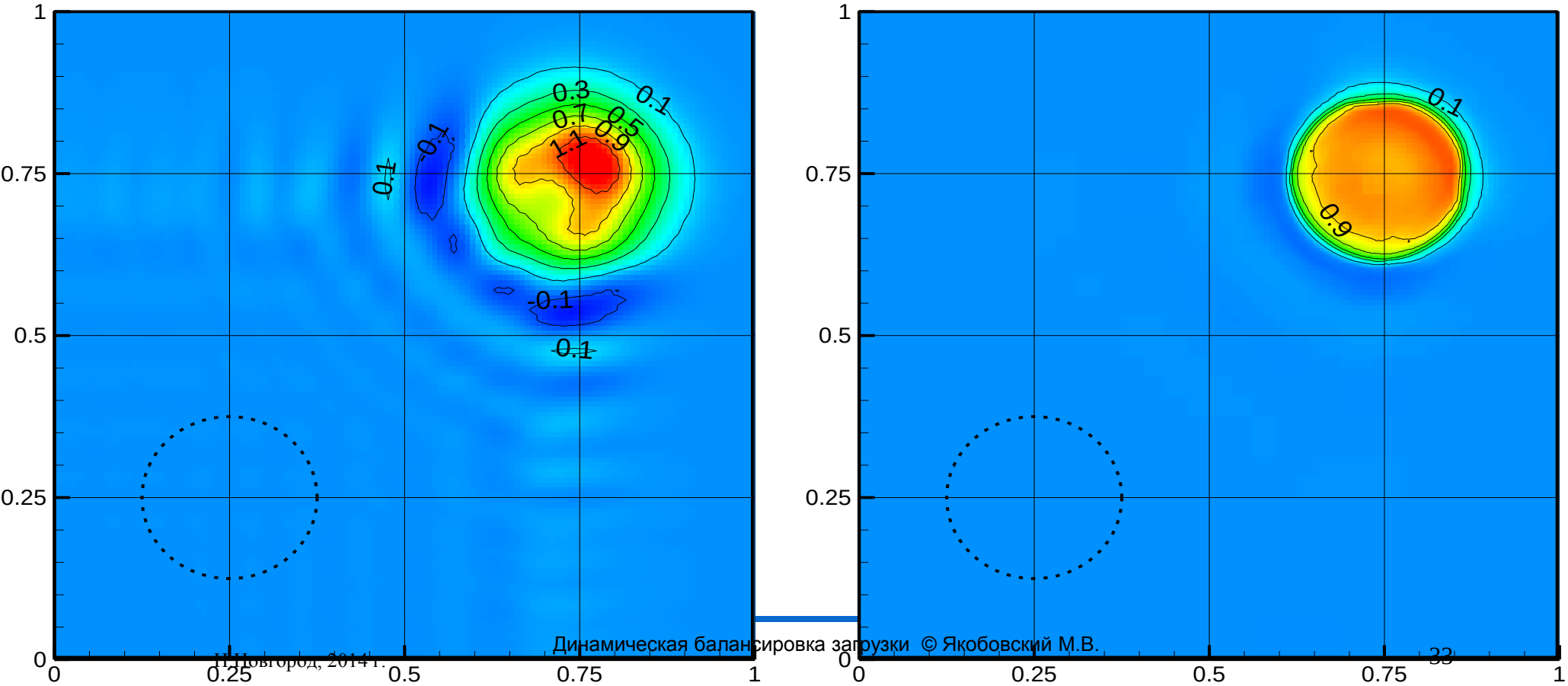


Дополнительные ограничения на размеры ячеек:

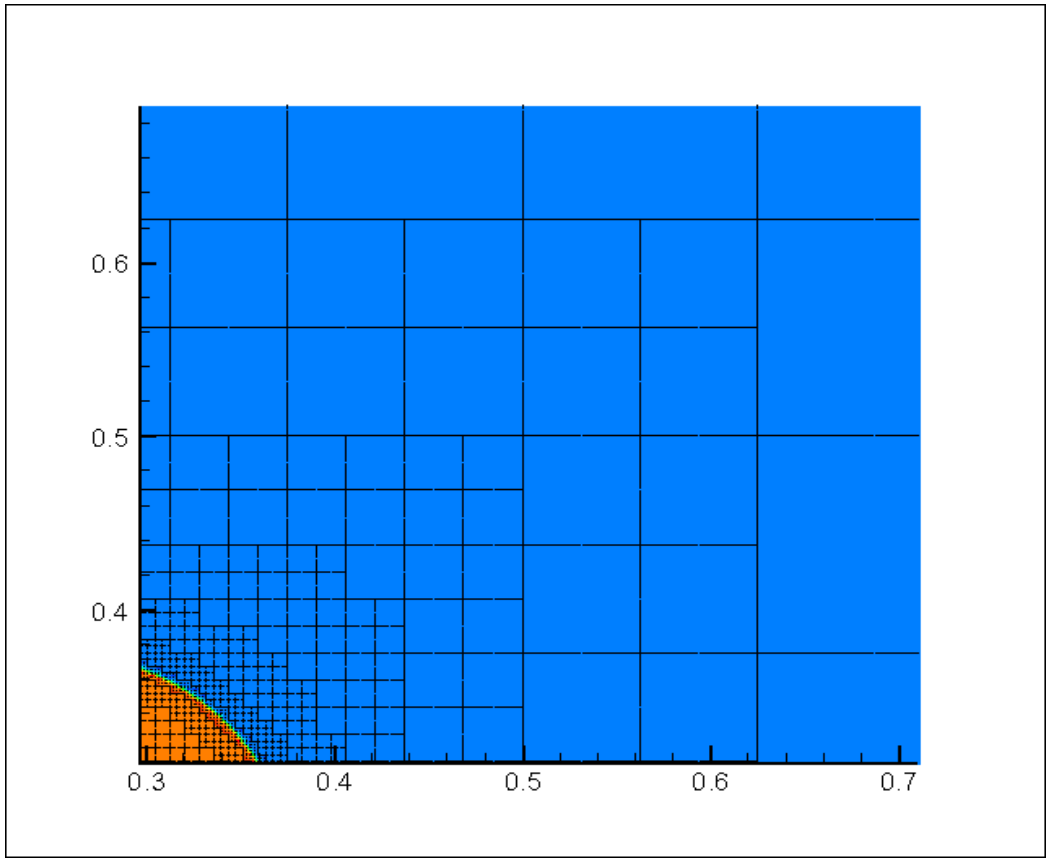
- Задан максимально допустимый размер ячеек
- Задан минимально допустимый размер ячеек
- Размеры соседних ячеек должны различаться не более, чем в 2 раза

Сравнение с равномерной сеткой

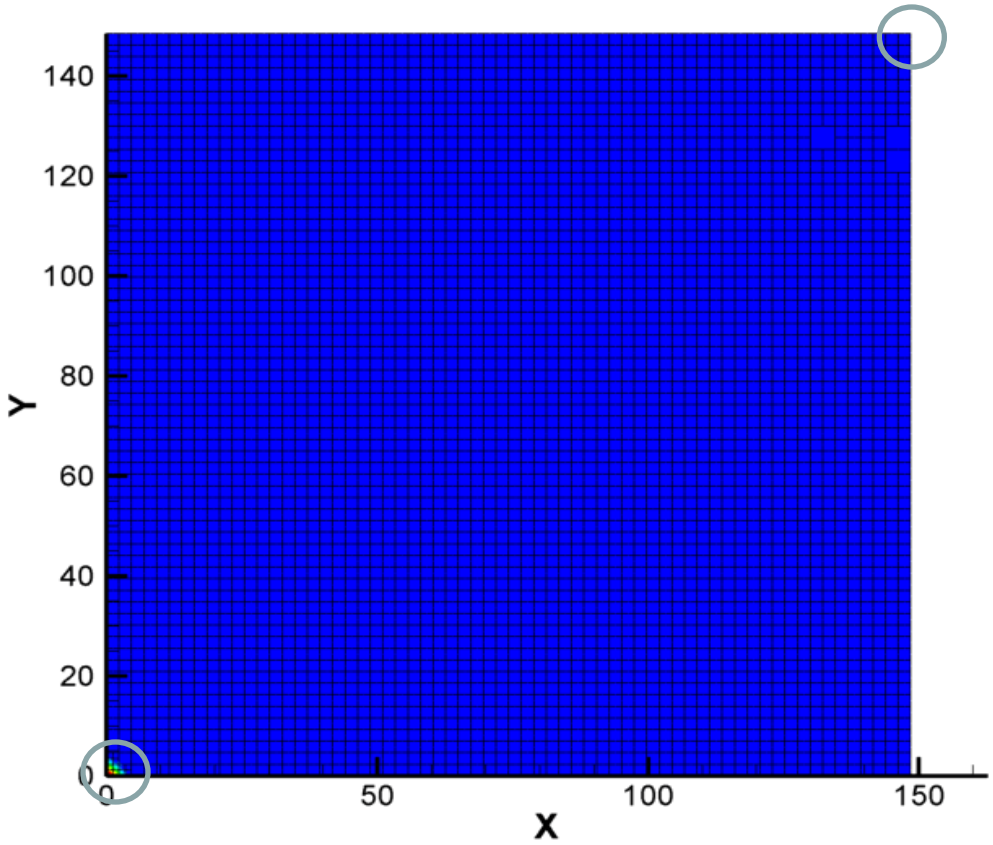
На рисунках показаны результаты решения простейшей задачи переноса на равномерной (слева) и адаптивной (справа) сетках с одинаковым числом ячеек (4096 штук). Скорость переноса направлена под углом 45° к линиям сетки; начальное условие показано пунктиром



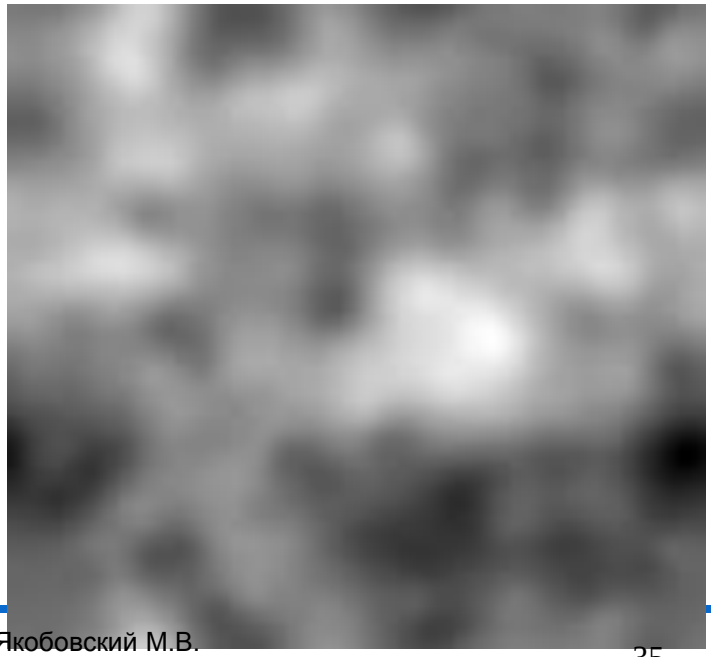
Адаптивная сетка



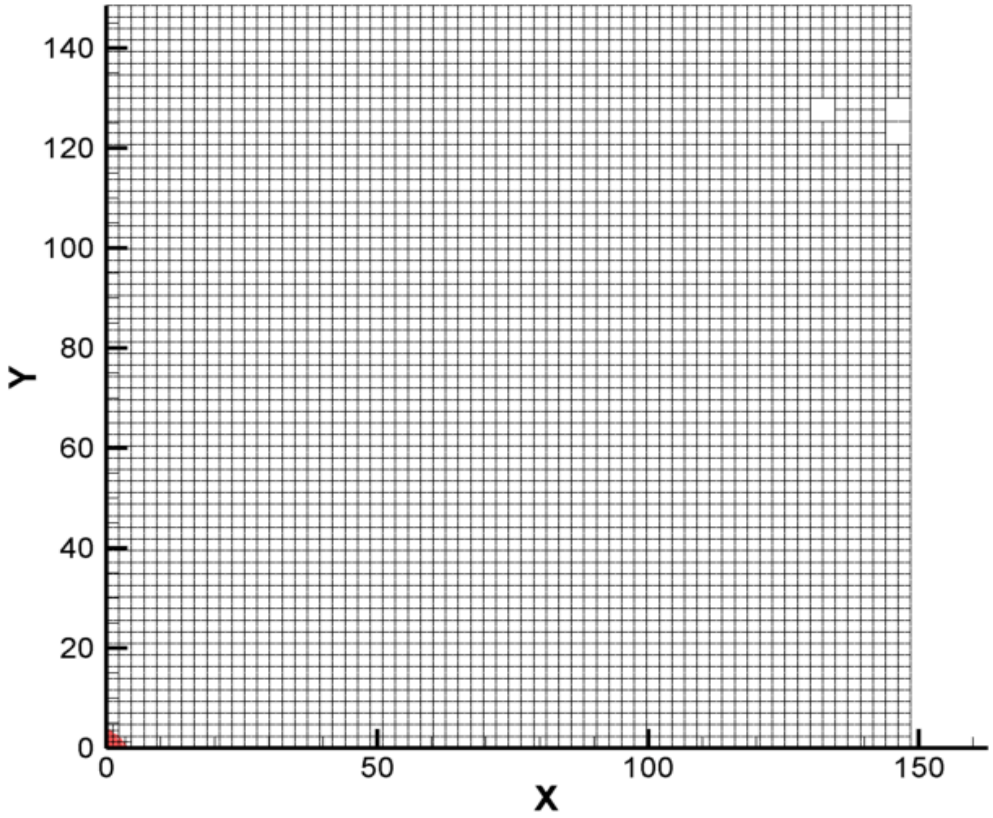
Решение двумерной задачи фильтрации нефтеводяной смеси в области с неоднородной проницаемостью



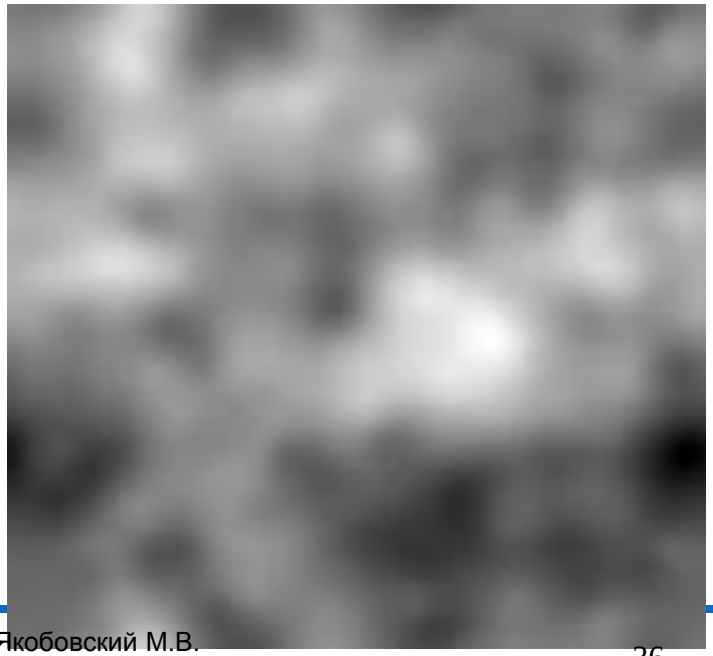
В юго-западном углу находится скважина, нагнетающая воду, в северо-восточном углу — добывающая скважина.
5-ти точечная схема
Поле проницаемости с разбросом значений на 4 порядка).



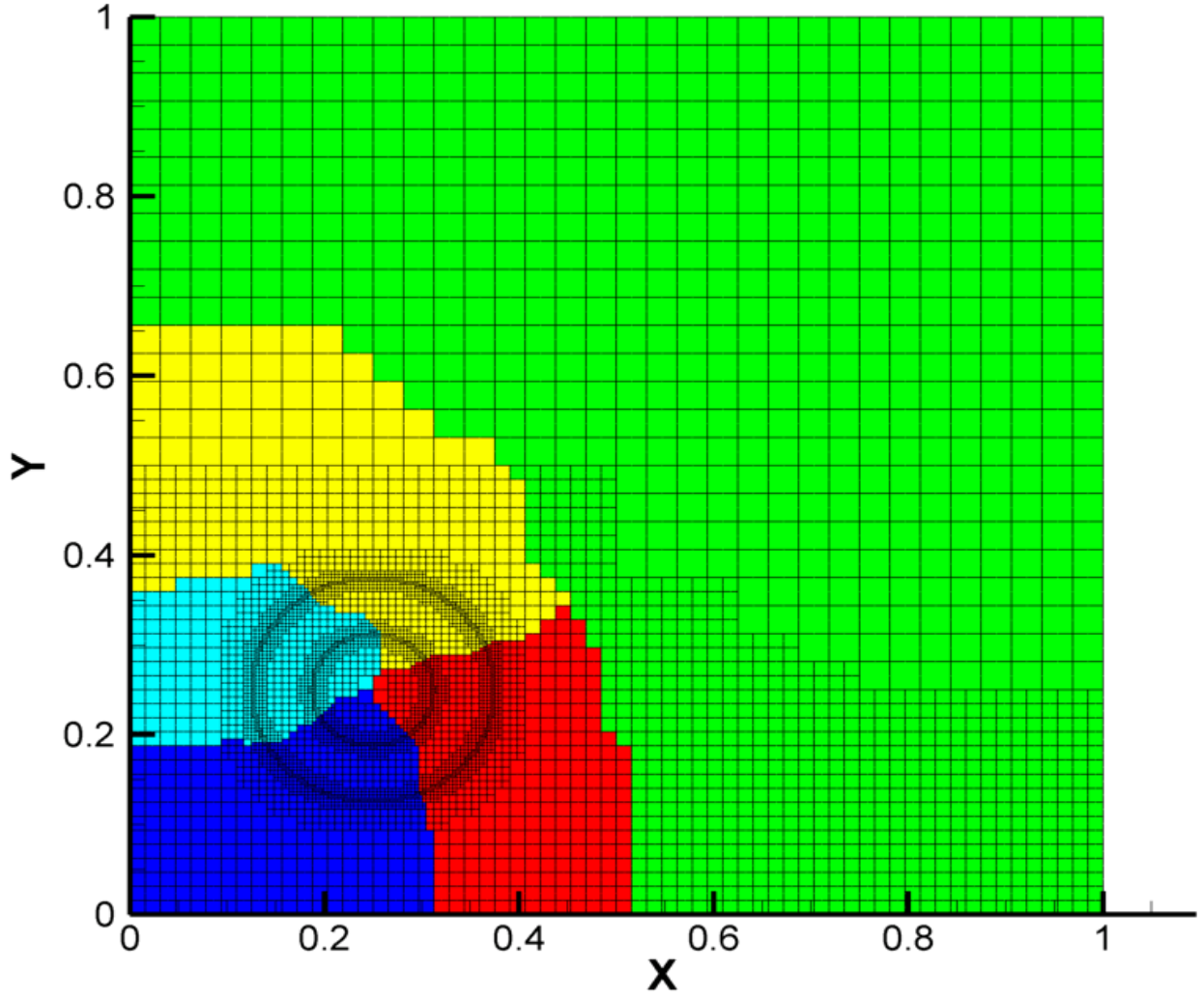
фильтрации нефтеводяной смеси в области с неоднородной проницаемостью

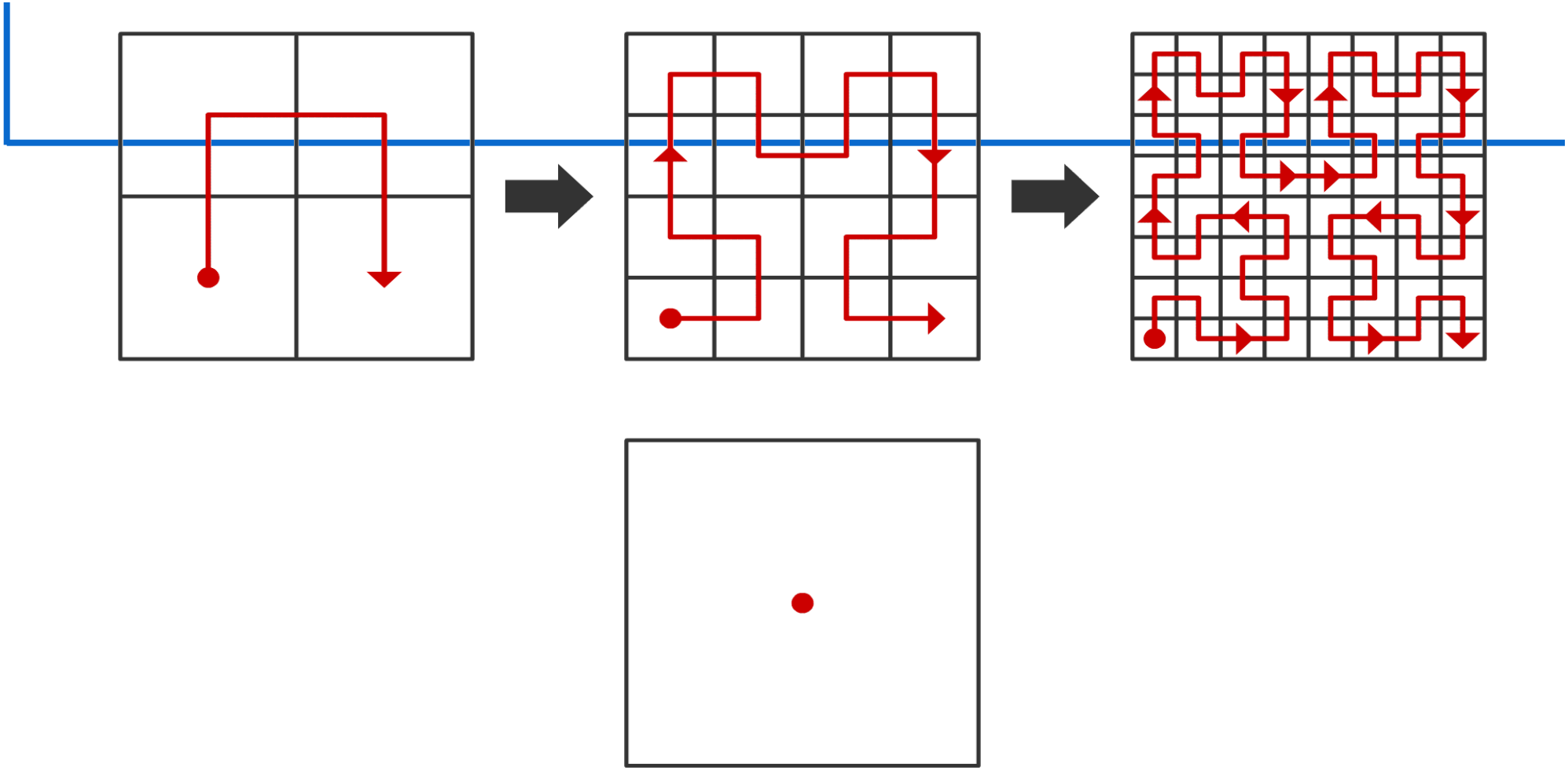


В юго-западном углу находится скважина, нагнетающая воду, в северо-восточном углу — добывающая скважина.
5-ти точечная схема
Поле проницаемости с разбросом значений на 4 порядка).



Декомпозиция пакетом Metis





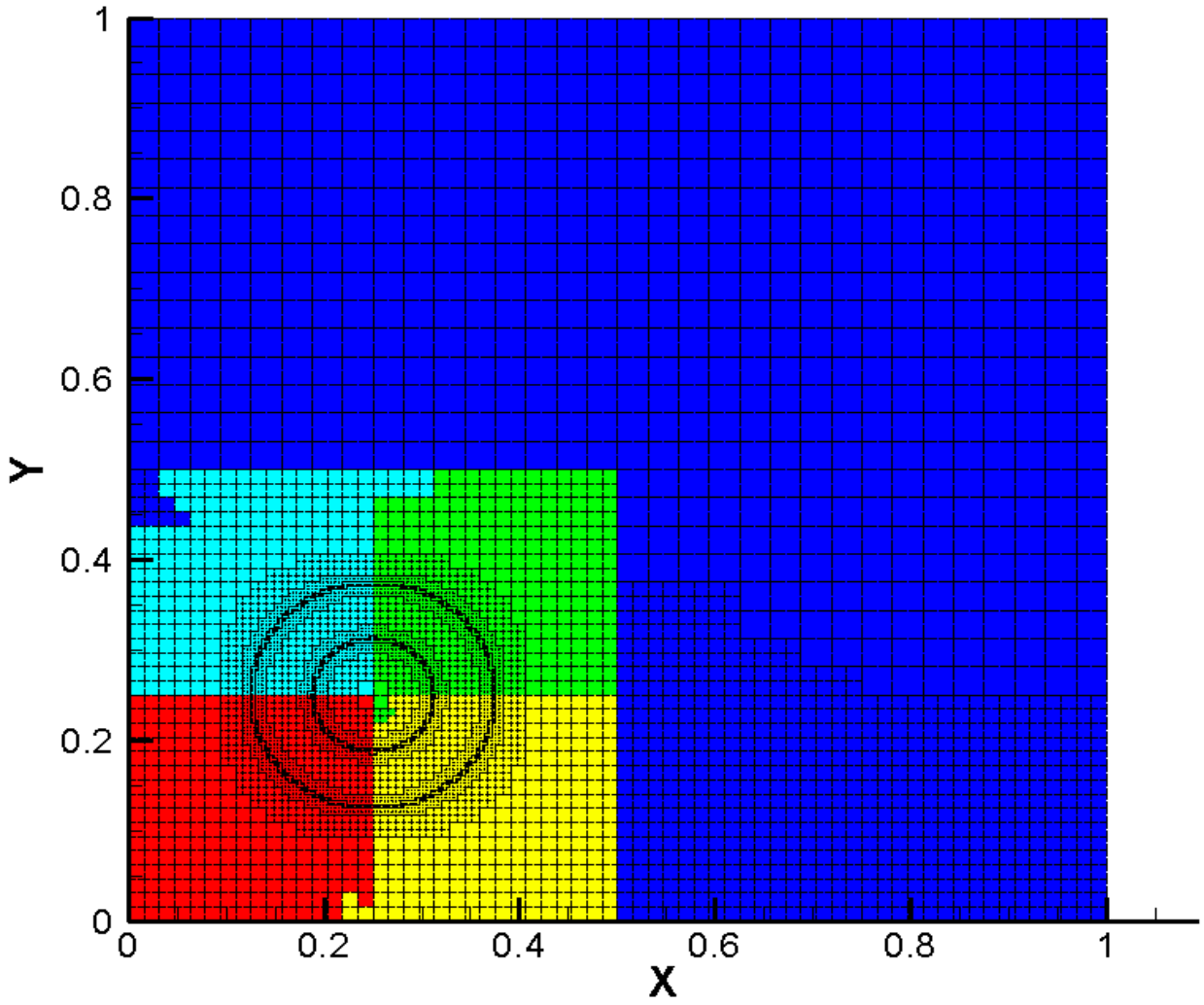
Hilbert-curve ordering

This ordering can be built by simple recursive procedure.

When mesh changes locally, Hilbert curve changes locally too.

It cannot be used for parallel computations due to chain dependence of elements.

Декомпозиция по кривой Гильберта

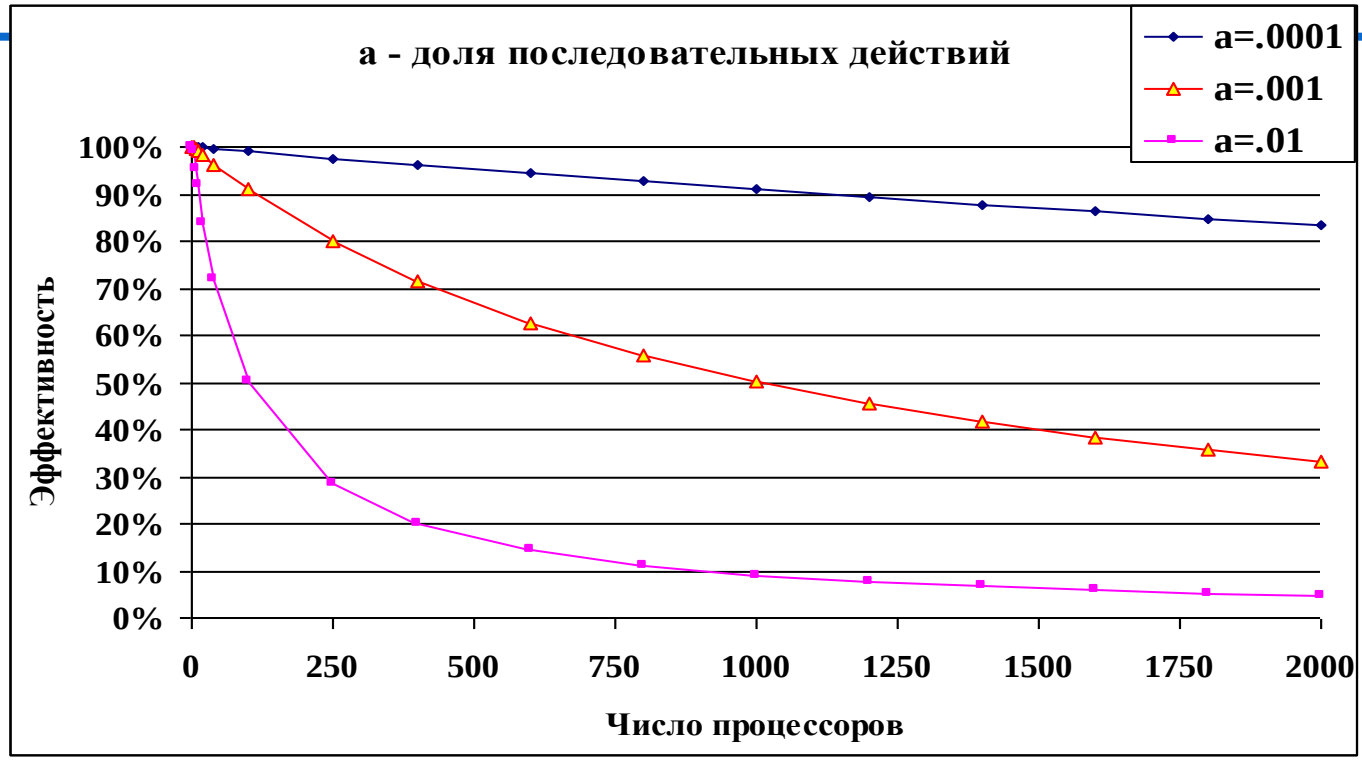


Ограничения

Закон Амдаля

$$S(p) = \frac{1}{a + \frac{1-a}{p}}$$

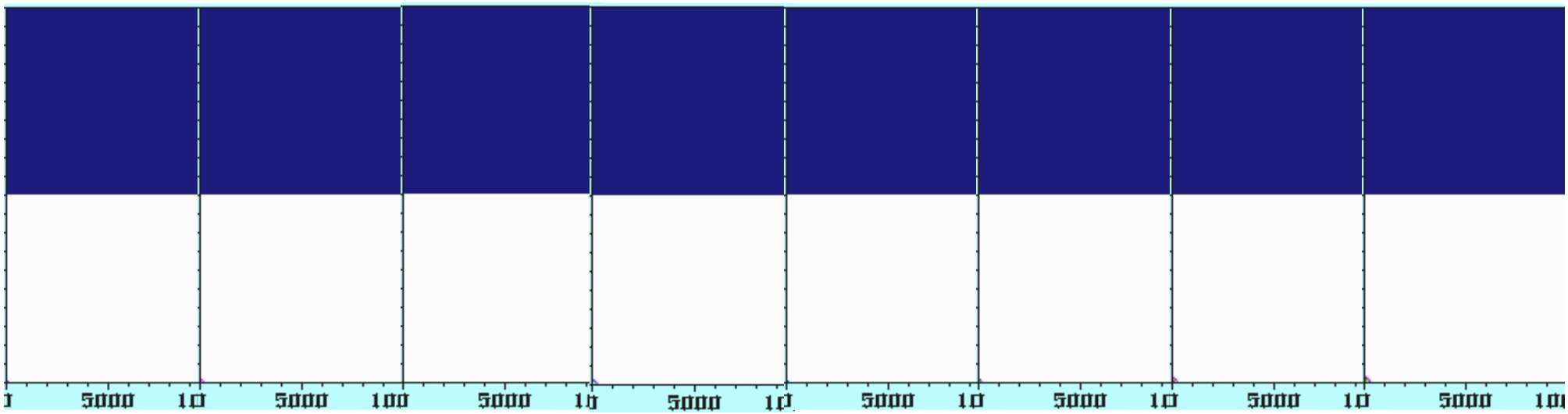
$$E(p) = \frac{1}{1 + a(p-1)}$$



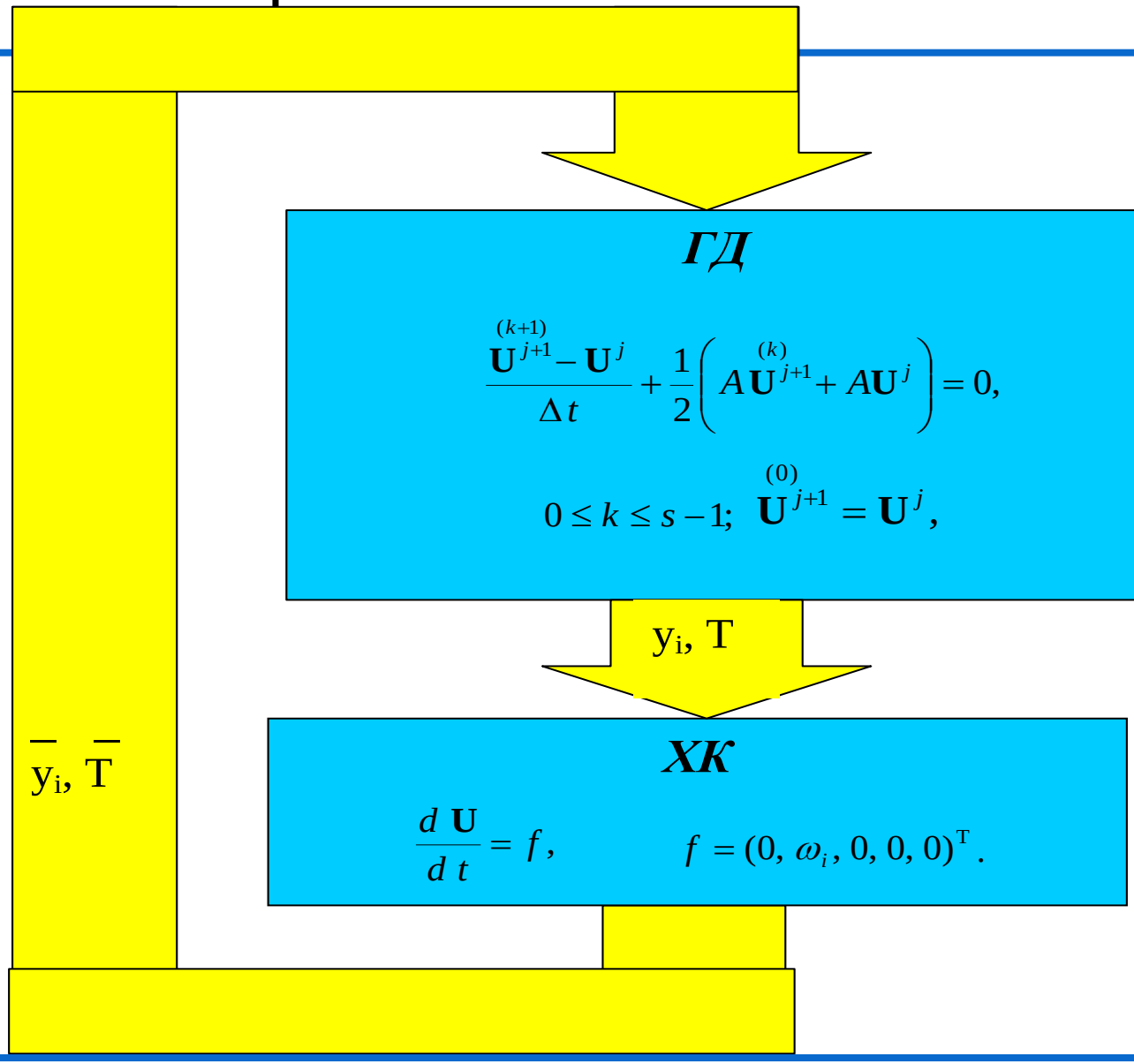
- Пакетный режим исполнения и отладки приложений
- Процедуры авторизованного доступа к удаленным системам
- Высокая динамика изменения конфигурации суперкомпьютеров
- Несоизмеримость ресурсов рабочей станции пользователя и суперкомпьютера

Methane combustion

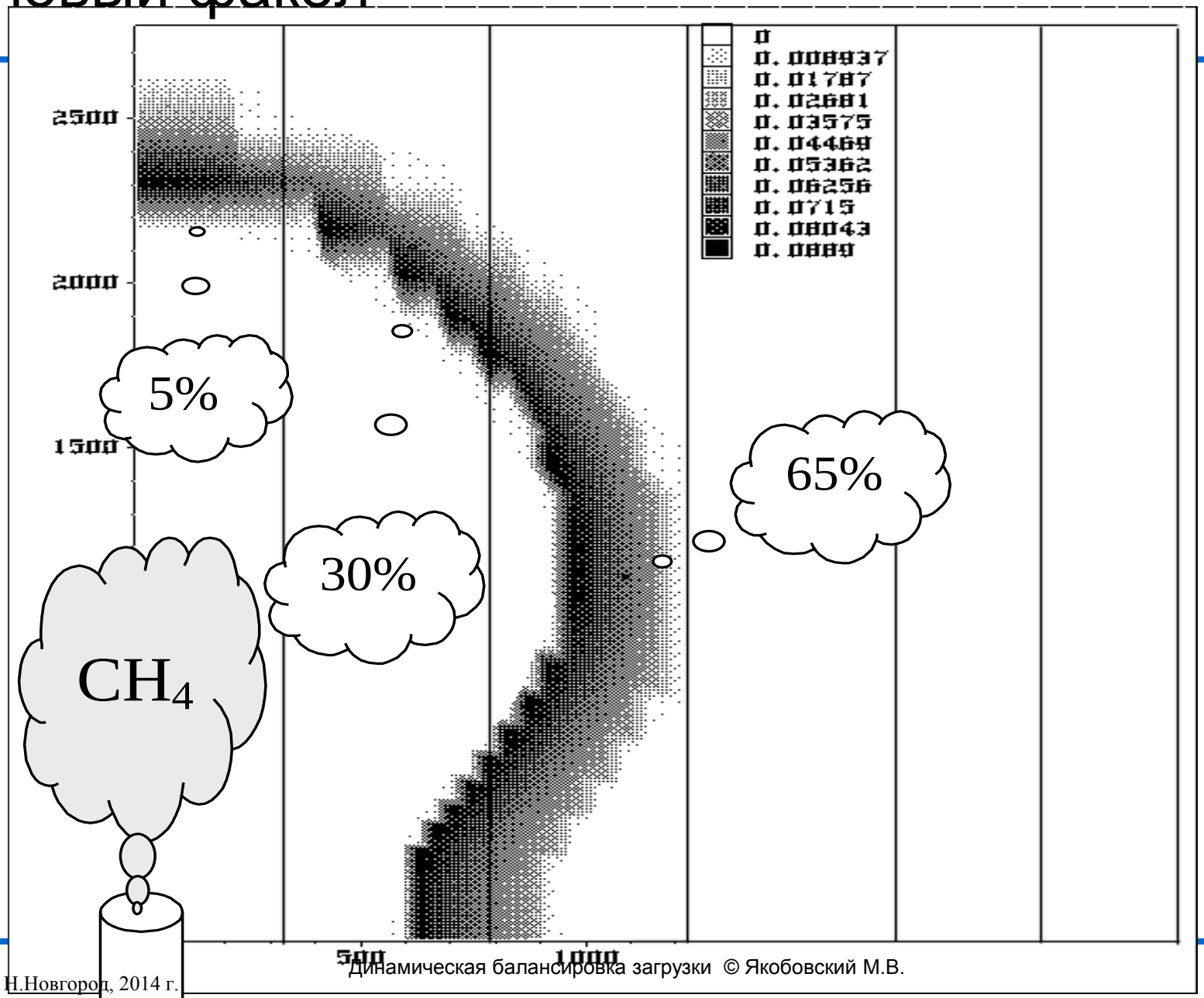
CH_4 NO NO_2 N_2O CH_3 CO CO_2 T



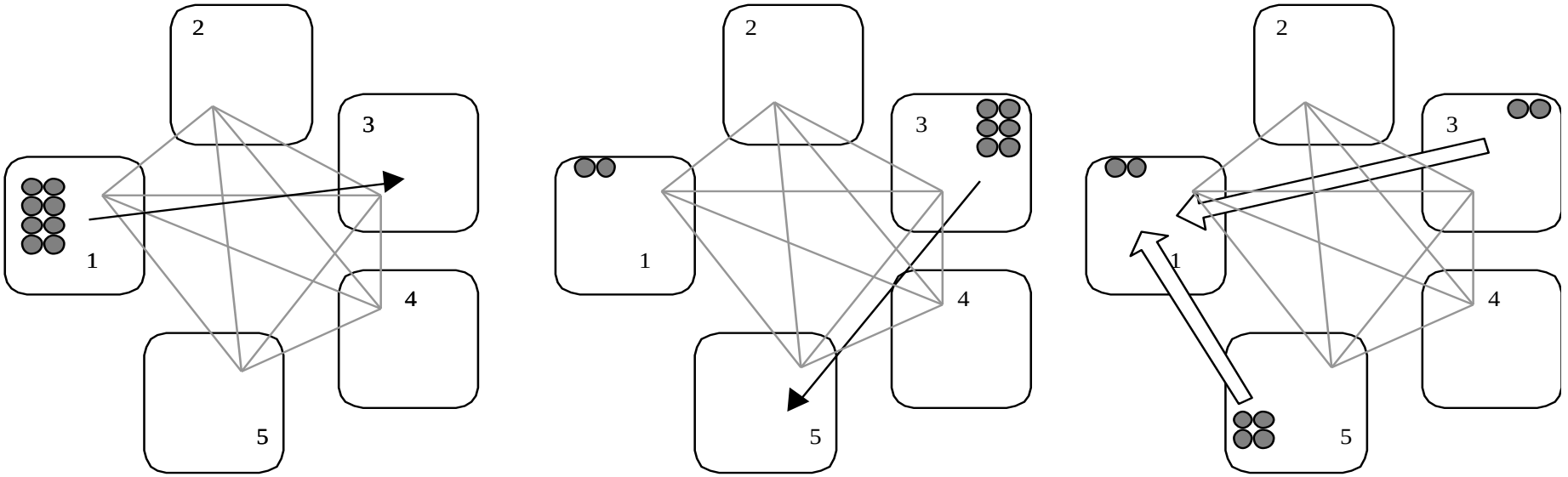
Блок схема алгоритма



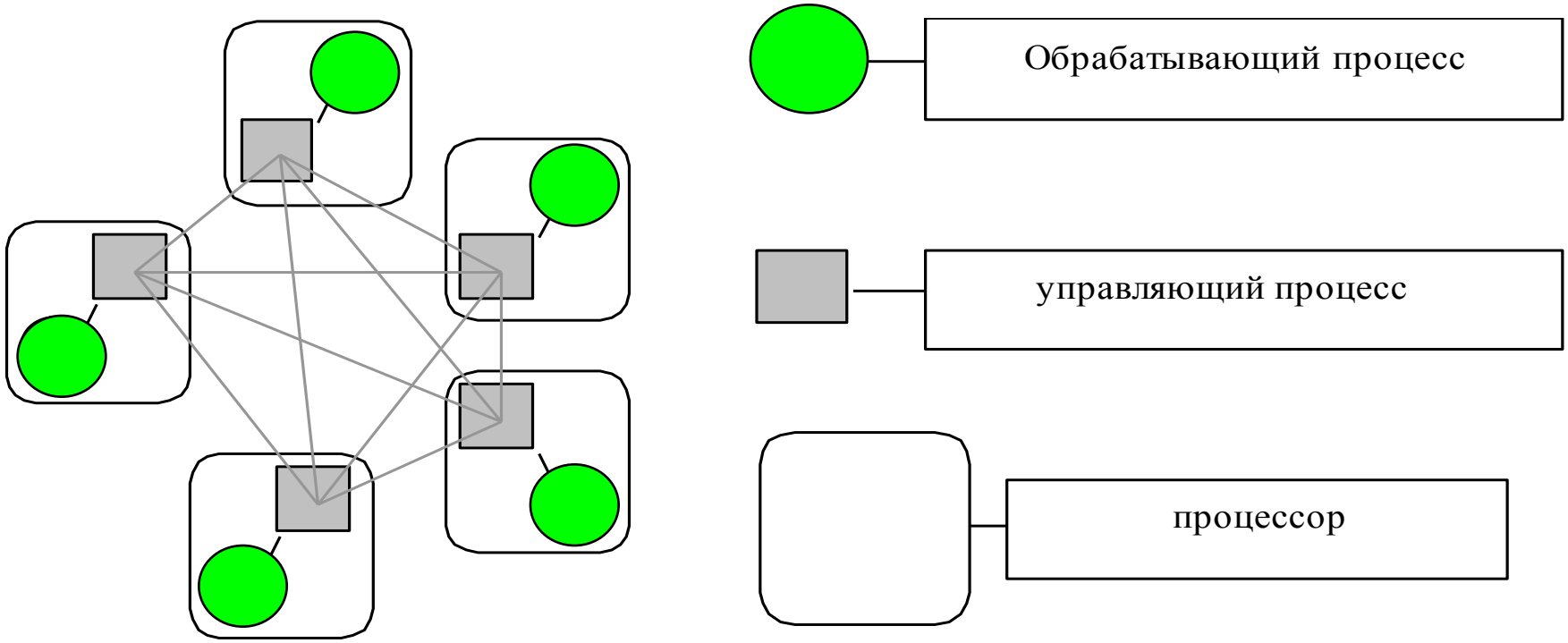
Метановый факел



Динамичекая балансировка



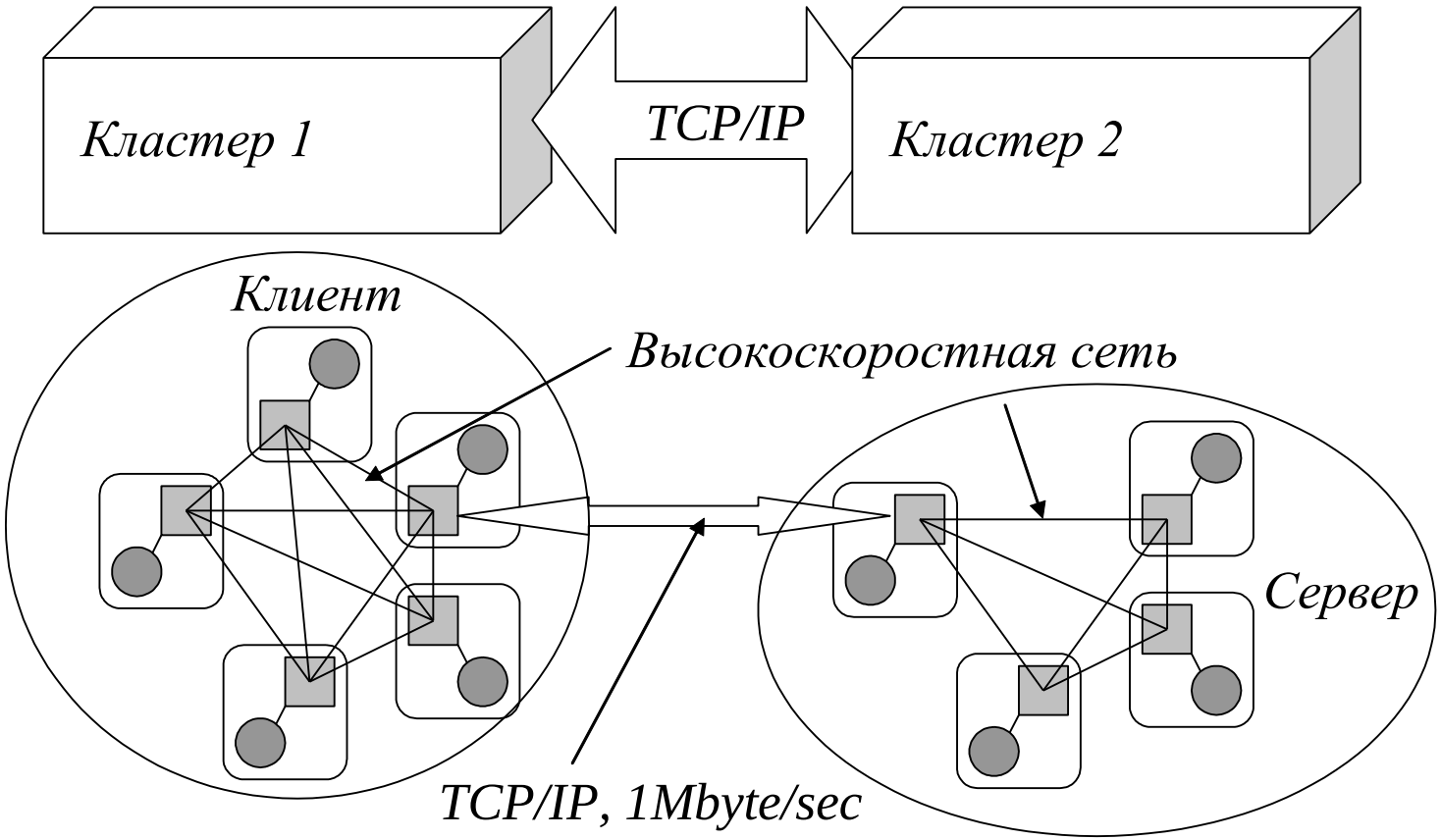
Основные процессы



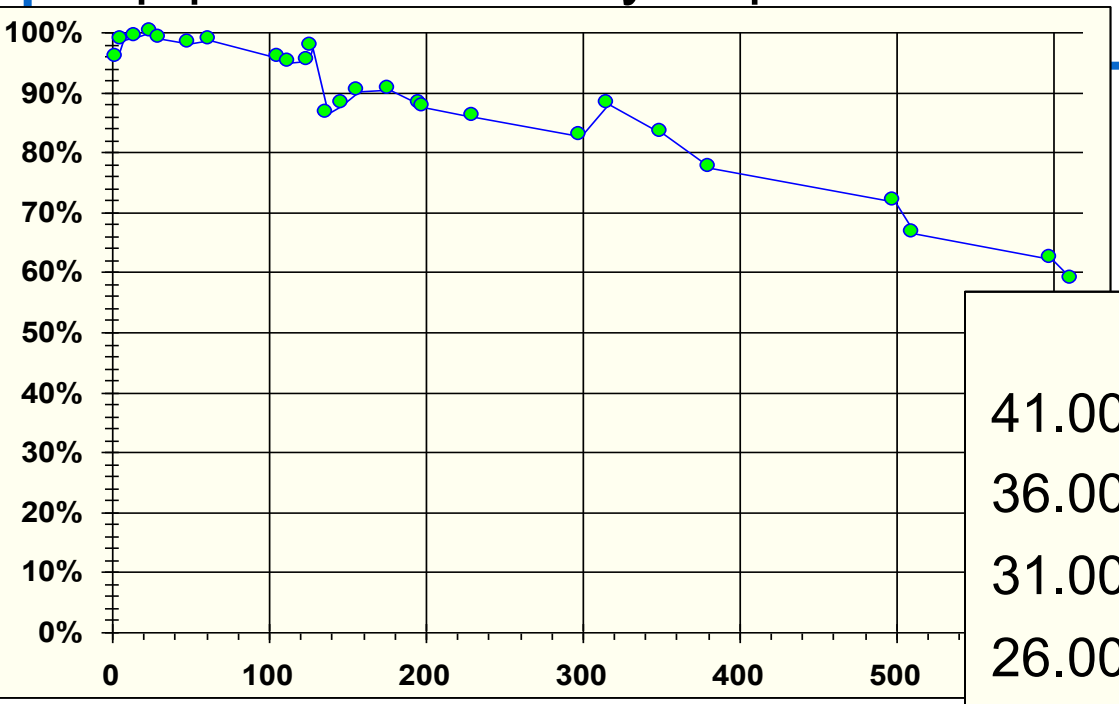
Завершение работы при выполнение всех условий

- нет локальных необработанных точек
- нет внешних точек
- нет обрабатываемых точек
- всем процессорам был послан запрос на получение необработанных точек
- всем процессорам было послано сообщение о том, что необработанные точки предоставлены быть не могут
- от всех процессоров получено сообщение о том, что необработанные точки предоставлены быть не могут
- все локальные точки обработаны и получены результаты обработки всех переданных точек

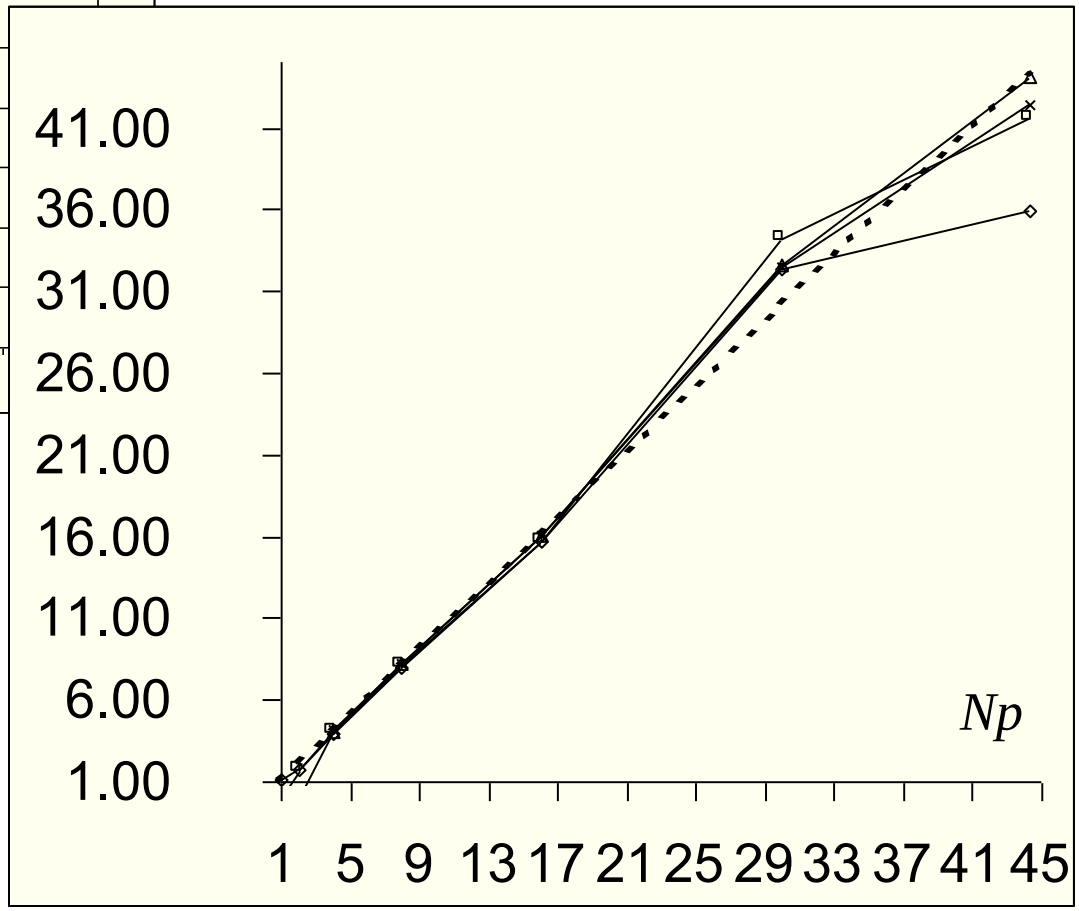
Структура программы при совместном использовании двух многопроцессорных комплексов



Эффективность и ускорение



сетка 1000x1000



Два кластера
33+12
процессоров

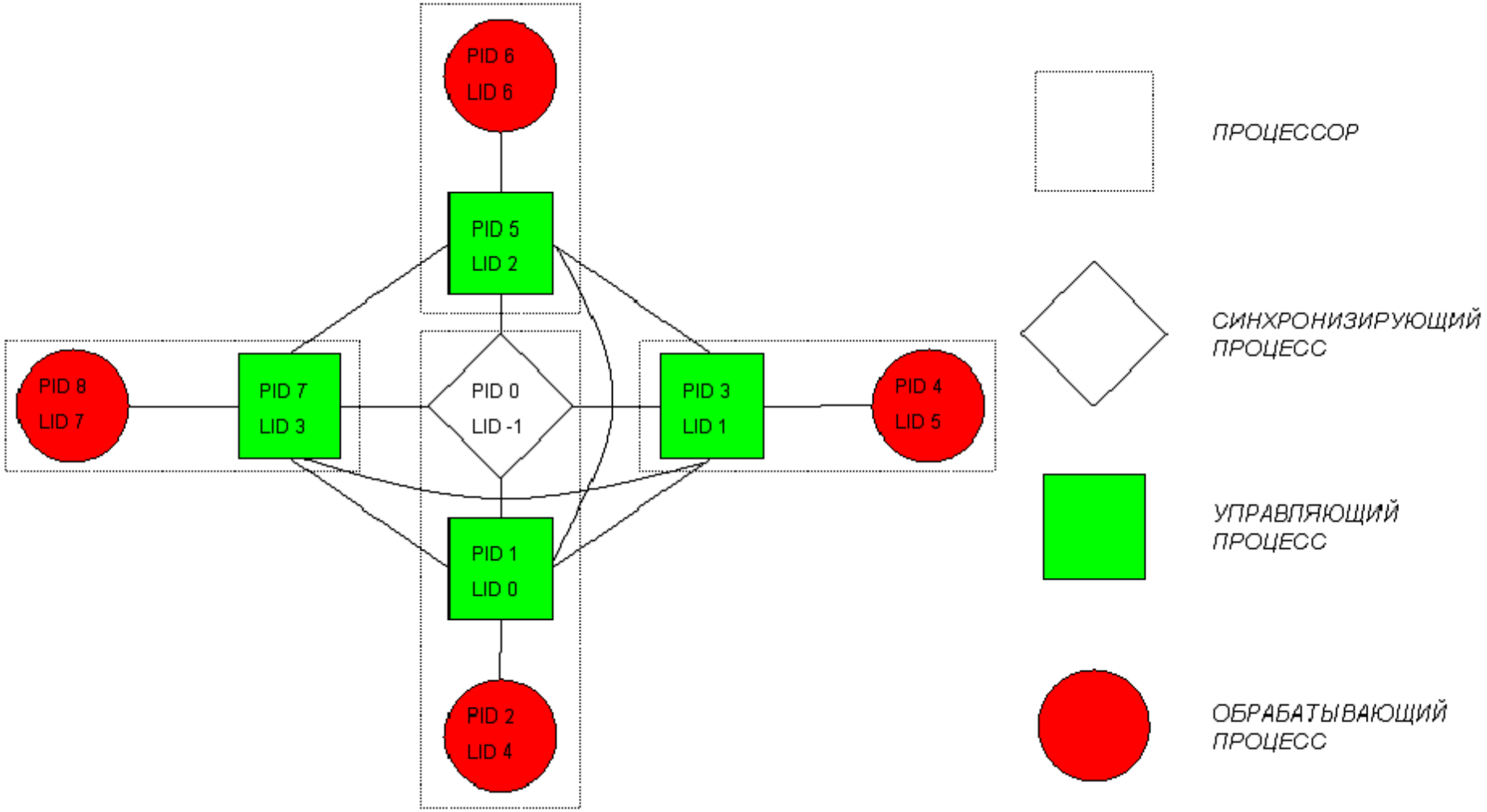
статическая балансировка

Число процессоров	Время	Ускорение
1	289	1.00
2	141	2.04
3	142	2.02
4	147	1.96
5	283	1.02
6	147	1.96
7	238	1.21
8	161	1.79

Динамическая балансировка загрузки

Число физических процессоров	Число процессов	Время	Ускорение	Эффективность
5	5(2)	158.18	1.8284865	91%
7	7(3)	107.5	2.6905116	90%
9	9(4)	82.95	3.4867993	87%
2	5(2)	113.85	2.540448	127%
3	7(3)	85.75	3.3729446	112%
4	9(4)	70.33	4.1124698	103%
5	11(5)	60.61	4.7719848	95%
6	13(6)	52.56	5.5028539	92%
7	15(7)	43.88	6.5913856	94%
8	17(8)	39.7	7.2853904	91%

Схема взаимодействия процессов



Типы процессов

- Тип процесса может быть определен с помощью функции **proc_type(void)**
- Синхронизирующий процесс T_MASTER обеспечивает согласованность выполнения расчета и записи результатов
- Управляющие процессы T_TASK отвечают за хранение данных, выполнение «диффузионного блока» и распределение заданий на этапе динамической балансировки загрузки.
- Обрабатывающие процессы T_CHEM выполняют обработку заданий из списков, формируемых на этапе динамической балансировки.

Серверный параллелизм

- Серверный параллелизм - метод динамической балансировки загрузки, применимый в случае обработки ***распределенного*** множества элементарных заданий ***непредсказуемой*** трудоёмкости

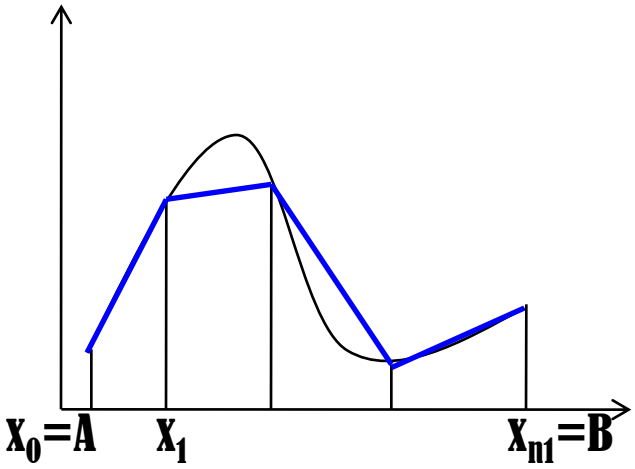
Параллельный алгоритм адаптивного интегрирования

- Метод динамической балансировки загрузки, применимый в случае обработки ***динамически*** порождаемого множества элементарных заданий ***непредсказуемой*** трудоёмкости

Постановка задачи

Вычислить с точностью ε значение определенного интеграла

$$J(A,B)=\int_A^B f(x)dx$$



Пусть на отрезке [*A,B*] задана равномерная сетка, содержащая *n+1* узел:

$$x_i = A + \frac{B-A}{n} i, \quad i = 0, \cdots, n$$

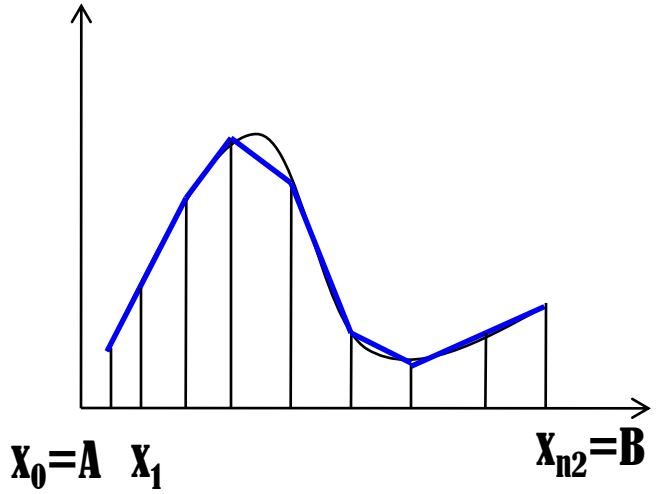
Тогда, согласно методу трапеций, можно численно найти определенный интеграл от функции на отрезке [A,B]:

$$J_n(A,B) = \frac{B-A}{n}\left(\frac{f(x_0)}{2} + \sum_{i=1}^{n-1} f(x_i) + \frac{f(x_n)}{2}\right)$$

Будем полагать значение *J* найденным с точностью ε, если выполнено условие:

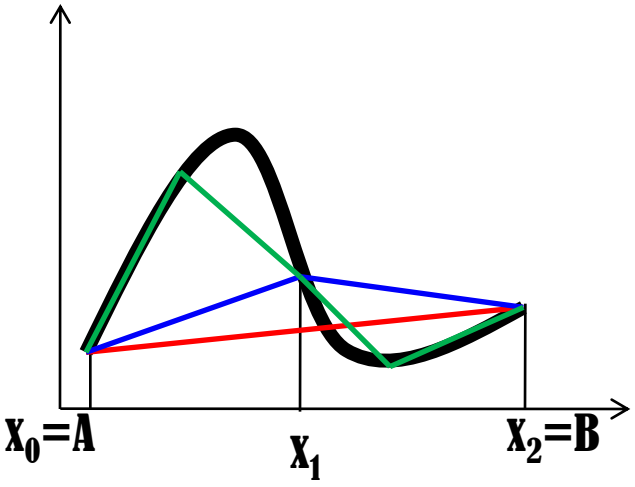
$$\left|J_{n1} - J_{n2}\right| \le \varepsilon \left|J_{n2}\right|$$

$$n_2 > n_1$$



Последовательный алгоритм интегрирования

```
IntTrap01(A,B)
{
n=1
J2n=(f(A)+f(B)) (B-A)/2

        do
        {
        Jn= J2n

        n=2n

        s=f(A)+f(B)

        for(i=1;i<n;i++)
           s+=2f(A+(B-A)i/n);

        J2n=s(B-A)/n;
        }
        while(|J2n- Jn|≥ε J2n)

return J2n
}
```



Недостатки:

в некоторых точках значение подынтегральной функции вычисляется более одного раза

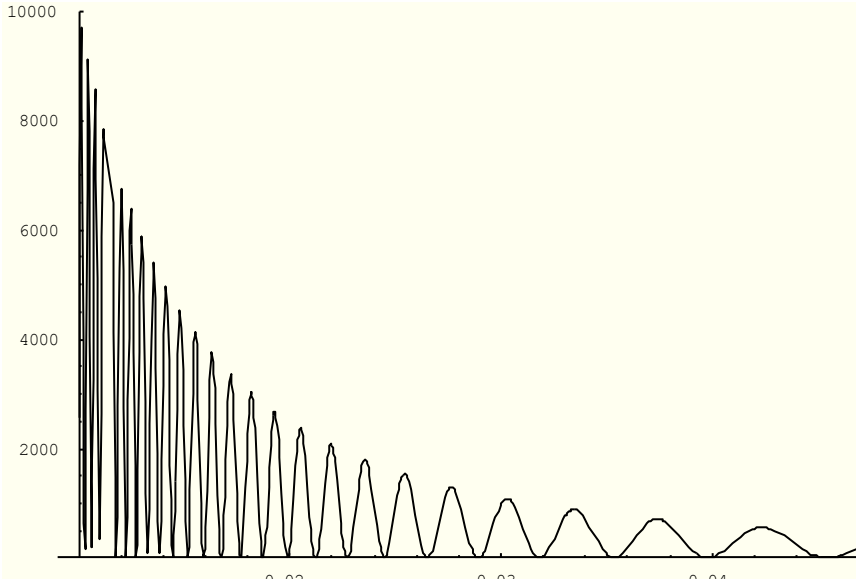
на всем интервале интегрирования используется равномерная сетка, тогда как число узлов сетки на единицу длины на разных участках интервала интегрирования, необходимое для достижения заданной точности, зависит от вида функции

Пример функции

$$f(x)=\frac{1}{x^2}\sin^2\left(\frac{1}{x}\right),\quad 0<A<<1$$

$$J(A,B)=\int_A^B \frac{1}{x^2}\sin^2\left(\frac{1}{x}\right)dx=-\frac{1}{2x}+\frac{1}{4}\sin\left(\frac{2}{x}\right)\Bigg|_A^B$$

$$J(A,B)=\frac{1}{4}\left(2\frac{B-A}{AB}+\sin\left(\frac{2}{B}\right)-\sin\left(\frac{2}{A}\right)\right)$$

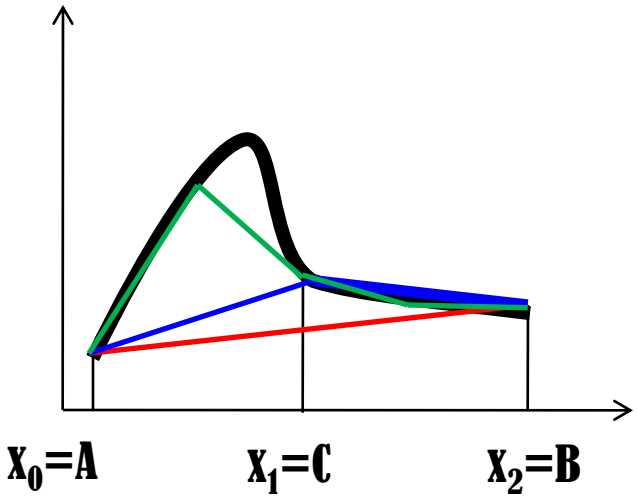


Результаты вычисления интеграла на разных отрезках

A	B	Npoints	eps real	time, c
0.00001	0.0001	1 553 568 289	-2.77E-11	434.55
0.0001	0.001	1 726 123 903	1.90E-10	470.99
0.001	0.01	360 075 831	2.05E-11	74.12
0.01	0.1	79 973 845	-2.22E-12	16.44
0.1	1	105 108 653	8.67E-11	21.42
1	10	396 149	-6.00E-11	0.094
10	100	412 331	-6.30E-11	0.096

Адаптивный алгоритм

```
main()
{
J= IntTrap03(  A, B, f(A),f(B)   )
}
```



Недостаток:

координаты концов отрезков хранятся в программном стеке процесса и фактически недоступны программисту

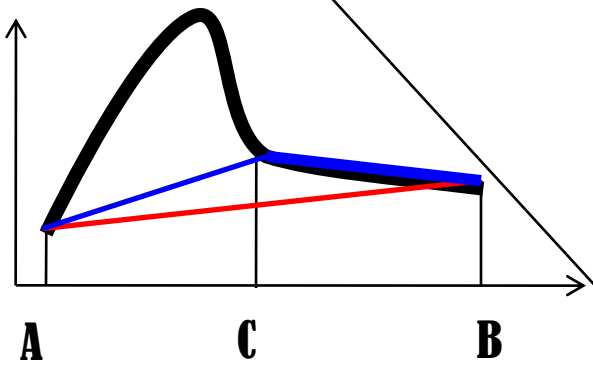
```
IntTrap03(A,B,fA,fB)
{
J=0

C=(A+B)/2

fC=f(C)

sAB=(fA+fB)*(B-A)/2

sAC=(fA+fC)*(C-A)/2
sCB=(fC+fB)*(B-C)/2

sACB=sAC+sCB

if(|sAB-sACB|≥ε|sACB|)

    J=IntTrap03(A,C,fA,fC)+IntTrap03(C,B,fC,fB)

else
    J=     sACB

return J
}
```

Преимущества:

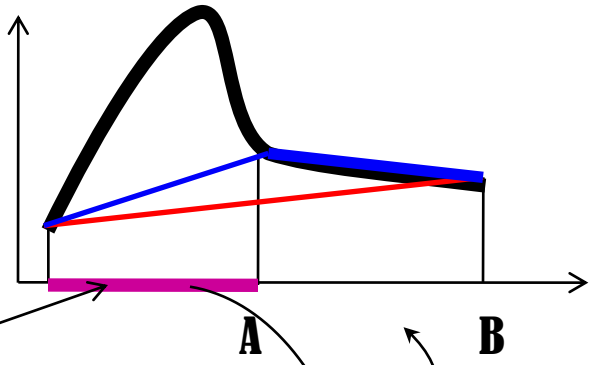
- нет повторных вычислений функции
- малый объём вычислений на гладких участках

Метод локального стека

```
IntTrap04(A,B)
{
J=0

fA=f(A)
fB=f(B)

sAB=(fA+fB)*(B-A)/2

while(1)
      {
      Тело цикла
      }
return J
}
```




```
C=(A+B)/2

fC=f(C)

sAC=(fA+fC)*(C-A)/2
sCB=(fC+fB)*(B-C)/2

sACB=sAC+sCB

if(|sAB-sACB|≥ε|sACB|)
      {
      PUT_INTO_STACK( A, C, fA, fC, sAC)
      A=C
      fA=fC
      sAB=sCB
      }
else
      {
      J+=sACB
      if(STACK_IS_NOT_FREE)
      break

      GET_FROM_STACK( A, B, fA, fB, sAB)
      }
```



Процедуры и данные обеспечивающие стек

```
// данные, описывающие стек

// указатель вершины стека
sp=0

// массив структур в которых
// хранятся отложенные задания
struct
    {
    A,B,fA,fB,s
    }
    stk[2000]
```

```
// макроопределения доступа к стеку

#define STACK_IS_NOT_FREE (sp>0)

#define PUT_INTO_STACK(A,B,fA,fB,s)
    {
    stk[sp].A=A
    stk[sp].B=B
    stk[sp].fA=fA
    stk[sp].fB=fB
    stk[sp].s=s
    sp++
    }

#define GET_FROM_STACK(A,B,fA,fB,s)
    {
    sp--
    A=stk[sp].A
    B=stk[sp].B
    fA=stk[sp].fA
    fB=stk[sp].fB
    s=stk[sp].s
    }
```

К вопросу о времени выполнения

- Тестирование показало, что при расчете с помощью алгоритма локального стека ***IntTrap04*** время работы было меньше, примерно на 5%, чем при использовании ***IntTrap03***

- Примем алгоритм ***IntTrap04*** за «наилучший» последовательный

Параллельный алгоритм интегрирования

- Метод геометрического параллелизма?
- Метод коллективного решения?
- ?

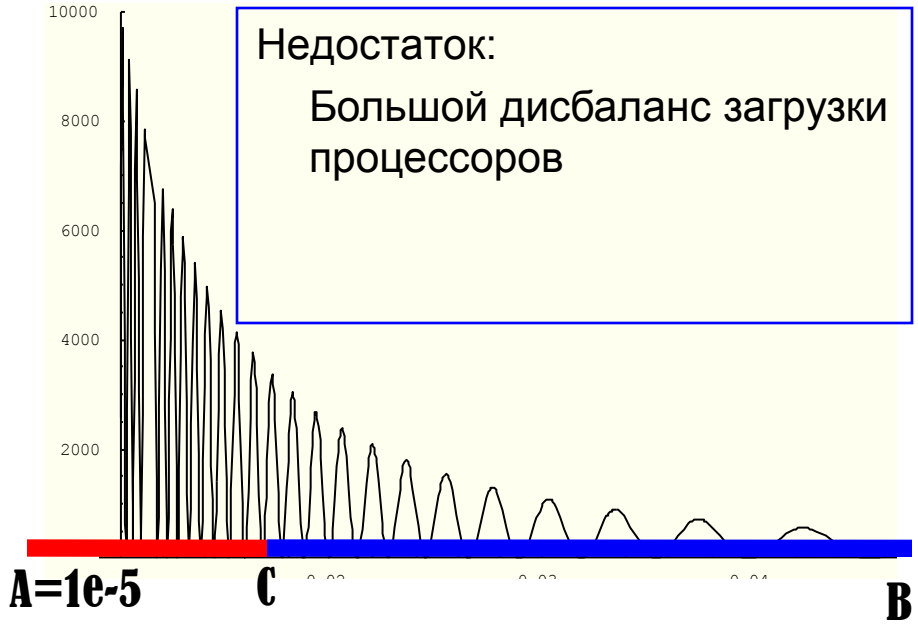
Метод геометрического параллелизма

```
main()
{
…
 for(i=0;i<p;i++)
    StartParallelProcess
           ( IntTrap04, A+(B-A)*i/p, A+(B-A)*(i+1)/p, &(s[i]) )

 WaitAllParallelProcess
 J=0

 for(i=0;i<p;i++)
    J+=s[i]

}
```

Недостаток:

Значительный дисбаланс загрузки процессоров

Расчет интеграла на разных отрезках



$$f(x)=\frac{1}{x^2}\sin^2\left(\frac{1}{x}\right)$$

p, ***(B-C)/ (C-A)***	интервал 1	интервал2	время1, с	время2, с
10	[1e-5, 0.10000900000]	[0.10000900000, 1]	**37.679**	**0.004**
100	[1e-5, 0.01000990000]	[0.01000990000, 1]	**37.274**	**0.037**
1 000	[1e-5, 0.00100999000]	[0.00100999000, 1]	**36.989**	**0.369**
10 000	[1e-5, 0.00010999900]	[0.00010999900, 1]	**34.064**	**3.364**
100 000	**[1e-5, 0.00001999990]**	**[0.00001999990, 1]**	**18.869**	**18.822**

Метод коллективного решения

```
main()
{
// Порождение p параллельных процессов,
// каждый из которых выполняет процедуру slave

for(k=0;k<p;k++)
     StartParallel(slave #k)

i=0 // число переданных для обработки интервалов
// n – число отрезков интегрирования

for(k=0;k<p;k++)
     { // Передача концов отрезков интегрирования
     Send(slave #k, A+(B-A)*i/n, A+(B-A)*(i+1)/n)
     i++
     }

// J – значение интеграла на всем интервале [A,B]
J=0
```

n - какое

Недостаток:

- Либо большой дисбаланс загрузки процессоров
- Либо большой объем лишних вычислений

```
Пока есть отрезки, не переданные для отработки,
следует дождаться сообщения от любого из процессов slave,
вычислившего  частичную сумму на переданном ему отрезке,
Получить значение этой суммы, прибавить к общему значению
Интеграла и передать освободившемуся процессу очередной
отрезок

while(i<n)
     {
     Recv(slave #any, s)
     J+=s
     Send(slave #any, A+(B-A)*i/n, A+(B-A)*(i+1)/n)
     i++
     }

// Получить результаты вычислений переданных отрезков
// и прибавить их к общей сумме

for(k=0;k<p;k++)
     {
     Recv(slave #any, s)
     J+=s
     }
}
```

```
slave()
{
подчиненный процесс, вычисляющий
     значение интеграла на отрезке
     [a,b]

while(1){  Recv(main,a,b)
           s=IntTrap04(a,b)
           Send(main,s)
        }
}
```

Вывод

Практически непригодны для решения поставленной задачи методы

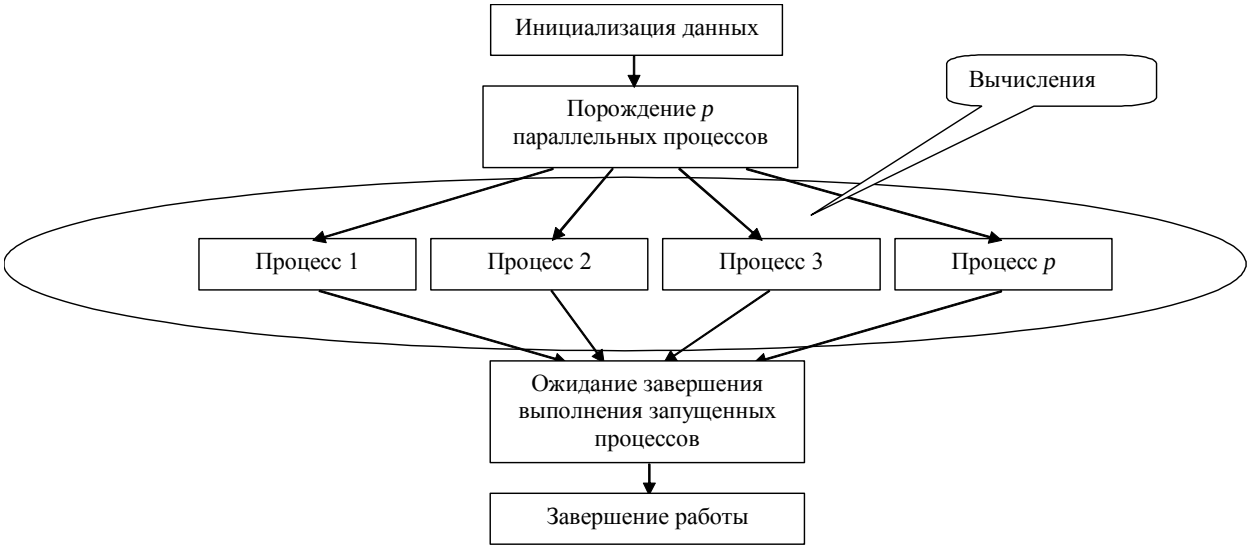
геометрического параллелизма (статическая балансировка)

и

коллективного решения (динамическая балансировка)

Метод глобального стека

- Вычислительные системы с общей памятью
- Динамическая балансировка загрузки
- Отсутствие централизованного управления



Идея алгоритма

- Пусть есть доступный всем параллельным процессам список отрезков интегрирования, организованный в виде стека. Назовем его **глобальным стеком**.
- Пусть у каждого процесса есть свой, доступный только этому процессу локальный стек
- Перед запуском параллельных процессов в глобальный стек помещается единственная запись (в дальнейшем "**отрезок**"):
  - координаты концов отрезка интегрирования,
  - значения функции на концах,
  - приближенное значение интеграла на этом отрезке.
- Тогда, предлагается следующая схема алгоритма, выполняемого каждым из параллельных процессов:

Пока в глобальном стеке есть отрезки:

- взять один **отрезок** из **глобального стека**

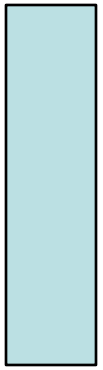
- выполнить алгоритм локального стека, но,

если при записи в **локальный стек** в нем есть несколько отрезков, а в глобальном стеке **отрезки** отсутствуют, то:

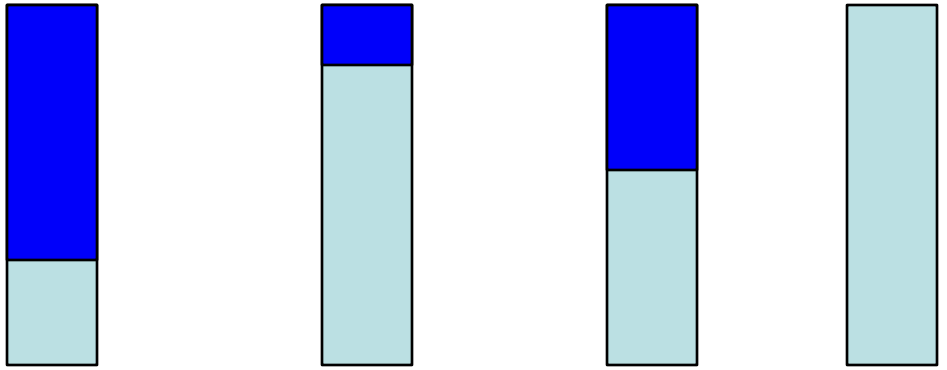
- переместить часть **отрезков** из **локального стека** в **глобальный стек**.

Основные структуры данных

- Глобальный стек

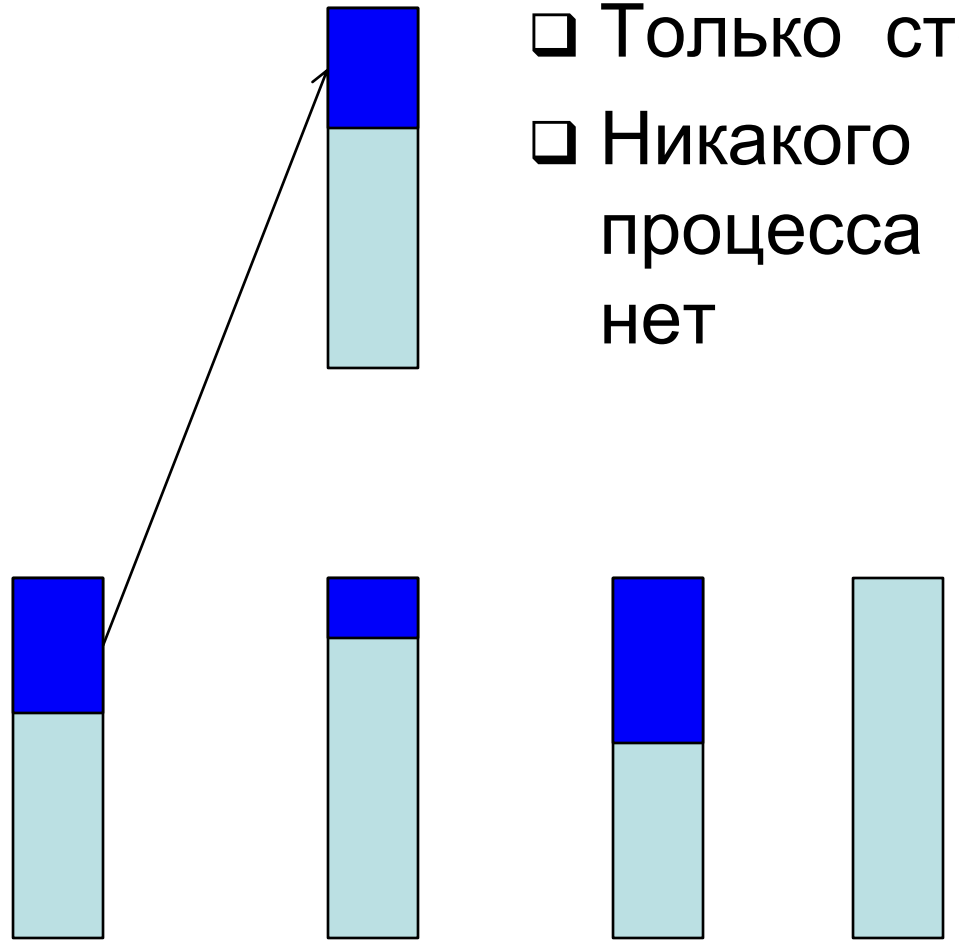


- Локальные стеки



Стеки алгоритма

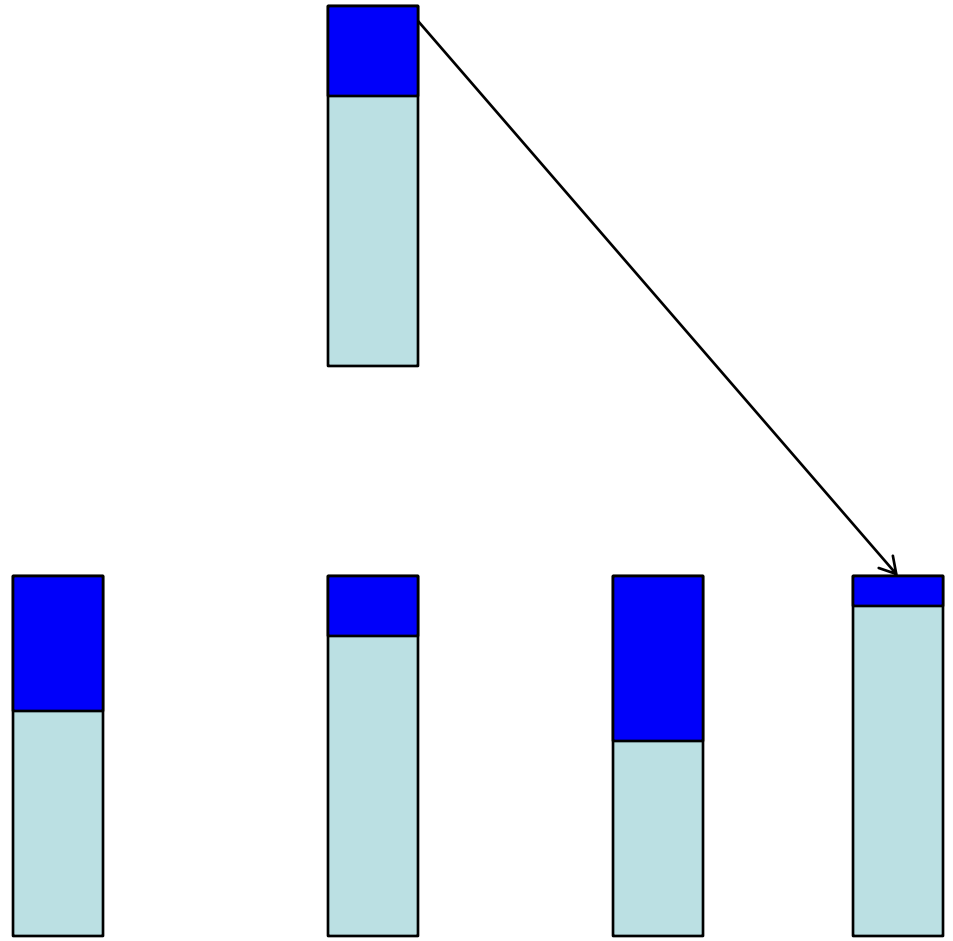
- Глобальный стек
- Локальные стеки



- Только стек
- Никакого процесса нет

Стеки алгоритма

- Глобальный стек
- Локальные стеки



Вопросы

- какую часть отрезков следует перемещать из локального стека в глобальный стек?
- в какой момент интеграл вычислен?
- что должен делать процесс у которого пуст локальный стек, если глобальный стек тоже пуст?
  - должен ли процесс закончить работу, если в его локальном и в глобальном стеке отрезков нет?

Преждевременный выход

- Интегрирующие процессы не должны заканчивать работу до тех пор, пока ***все*** отрезки интервала интегрирования не будут полностью обработаны
- Преждевременное завершение работы приведет к получению верного ответа, но на меньшем числе процессоров и за большее время

Результаты тестирования

$$J = \int_{10^{-5}}^{1} \frac{1}{x^2} \sin^2\left(\frac{1}{x}\right)$$

Время выполнения

Np	*1*	*2*	*3*	*4*
tiger.jscc.ru	*31.39*	*15.61*	*10.29*	*7.83*
ga03.imamod.ru	*37.48*	*19.00*	*-*	*-*

Ускорение

Np	*1*	*2*	*3*	*4*
tiger.jscc.ru	*1*	*2.01*	*3.05*	*4.01*
ga03.imamod.ru	*1*	*1.97*		

Эффективность

np	*1*	*2*	*3*	*4*
tiger.jscc.ru	*100%*	*101%*	*102%*	*100%*
ga03.imamod.ru	*100%*	*99%*		

Литература…

- Якобовский М.В. Введение в параллельные методы решения задач: Учебное пособие / Предисл.: В. А. Садовничий. – М.: Издательство Московского университета, 2013. – 328 с., илл. – (Серия «Суперкомпьютерное образование») ISBN 978-5-211-06382-2
- *Лацис А.О.* Параллельная обработка данных. – М.: Академия, 2010 г. 336 стр. ISBN 978-5-7695-5951-8
- *Якобовский М.В., Кулькова Е.Ю.* Решение задач на многопроцессорных вычислительных системах с разделяемой памятью. - М.: СТАНКИН, 2004. – 30 с. http://www.imamod.ru/~serge/arc/stud/Jackob_2.pdf

Литература

Ресурсы Internet

- http://parallel.ru
- http://top500.org
- http://supercomputers.ru

Контакты

Якобовский М.В.

проф., д.ф.-м.н.,

зав. сектором

«Программного обеспечения многопроцессорных систем и вычислительных сетей»

Института прикладной математики им. М.В.Келдыша Российской академии наук

mail: lira@imamod.ru

web: http://lira.imamod.ru