Пять парадигм параллельного программирования

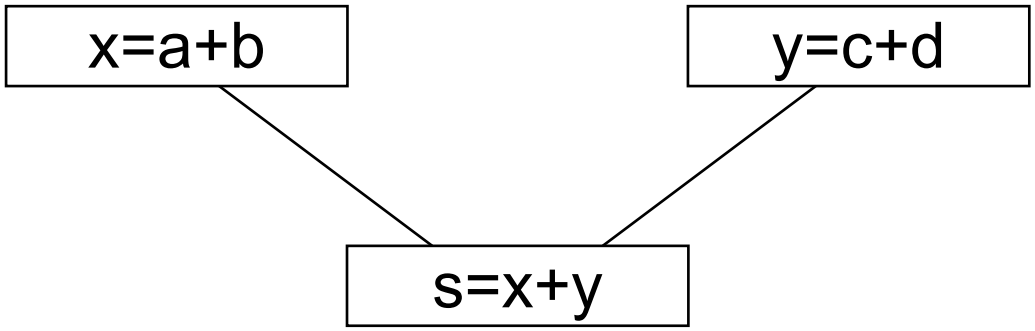
Хусаинов Ахмет Аксанович
husainov51@yandex.ru
http://husainov51.narod.ru

Основные разделы

1) **Что такое параллельная программа?**
2) **Пять стилей параллельного программирования**
3) **Ноутбук как симметричная мультипроцессорная система (SMP)**
4) **Как писать параллельные программы?**
5) **Итеративный параллелизм и семафоры**
6) **Математическое моделирование параллельных вычислительных систем**
7) **Моноиды трасс и вычислительные процессы**
8) **Полукубические множества**
9) **Моноиды трасс и полукубические множества**
10) **Рекурсивный параллелизм**
11) **Конвейерные системы**
12) **Производители и потребители: каналы**
13) **Клиент-сервер: задача о читателях и писателях**
14) **Асинхронные системы**
15) **Асинхронные системы и кубические множества**
16) **Взаимодействующие каналы**
17) **Сети Петри и асинхронные системы**
18) **Топология – «резиновая» геометрия**
19) **Числа Бетти**
20) **Вычисление чисел Бетти**
21) **Числа Бетти полукубических множеств**
22) **Числа Бетти асинхронных систем**
23) **Числа Бетти сетей Петри**
24) **Открытые проблемы**

Что такое параллельная программа?

- Время вычисления суммы s=((a+b)+c)+d равно 3 такта
- Для двух *параллельно работающих процессоров* существует программа, вычисляющая за 2 такта



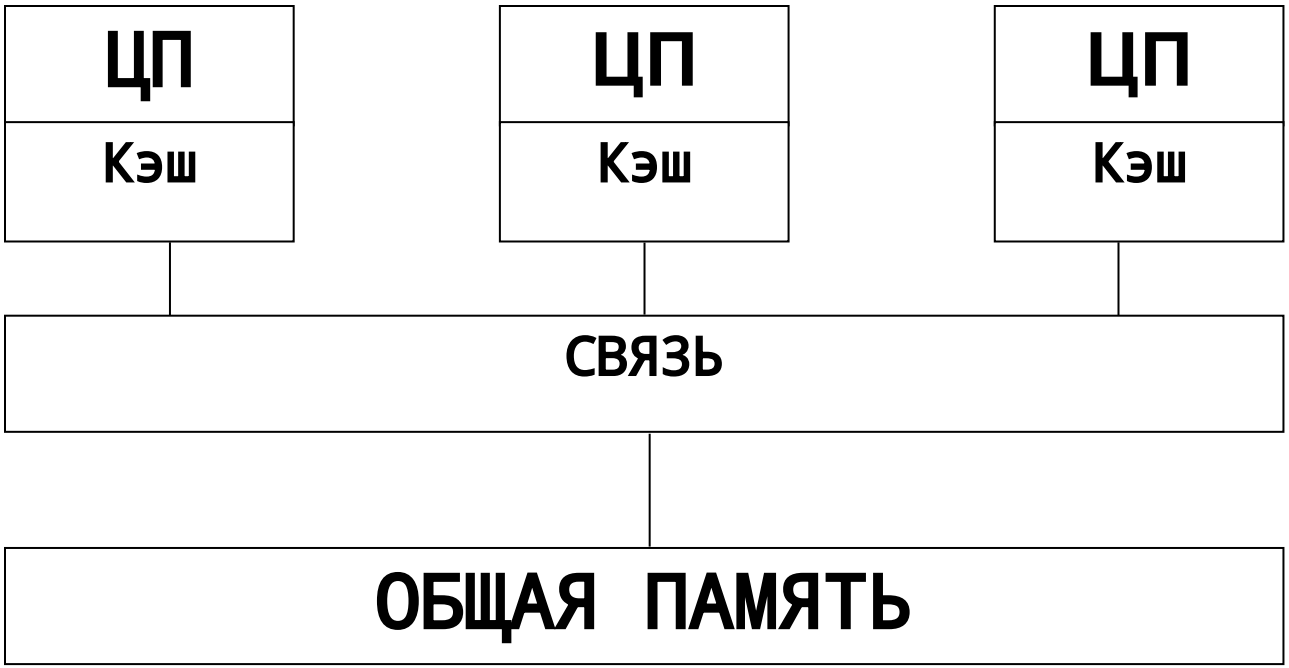
Пять стилей параллельного программирования

- Итеративный параллелизм
- Рекурсивный параллелизм
- Производители и потребители
- Клиенты и серверы
- Взаимодействующие каналы (или взаимодействующие равные)

Эндрюс Г.Р. Основы многопоточного, параллельного и распределенного программировния. – М.: Изд. дом «Вильямс», 2003

Ноутбук как симметричная мультипроцессорная система (SMP)

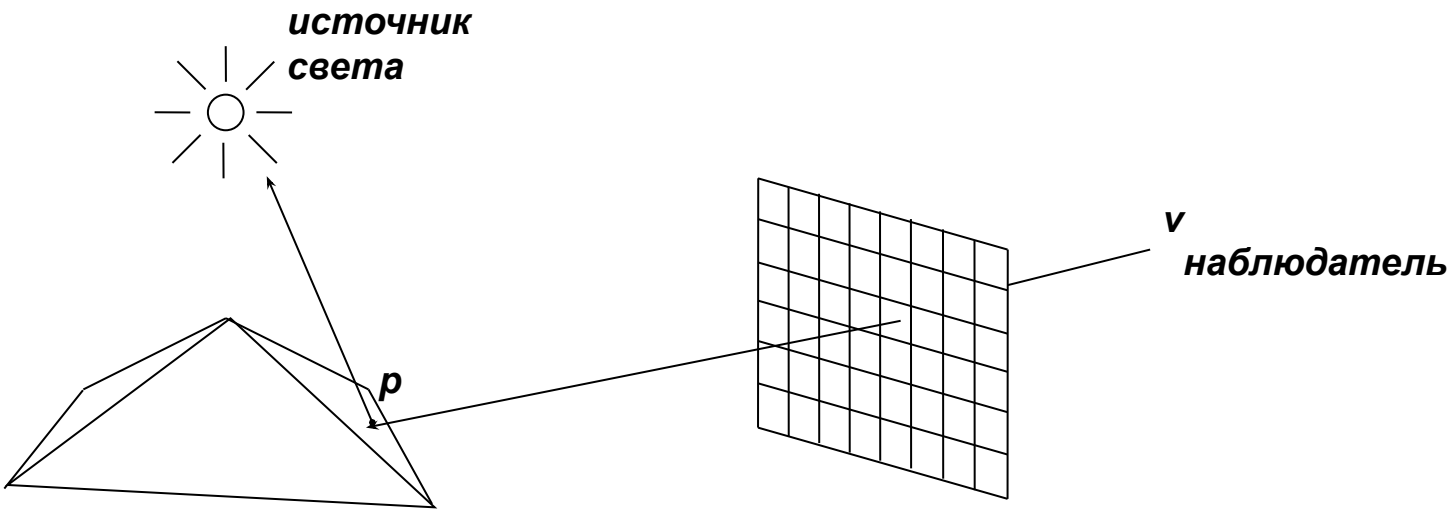
Архитектура SMP:



Как писать параллельные программы?

- **1. Разрабатываются подпрограммы, которые могут выполняться независимо. Например, для *метода трассировки лучей* пишется подпрограмма**

DWORD WINAPI Part(void *a) {…}

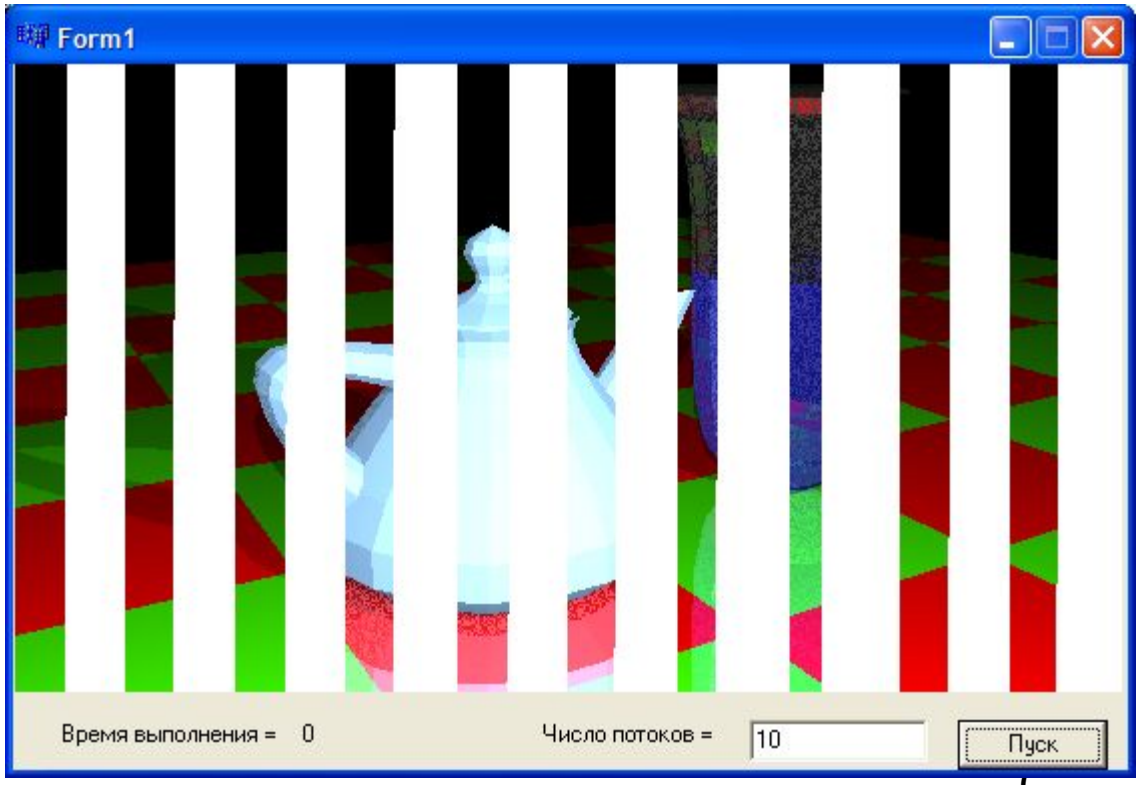


выводящая точки в заданную прямоугольную область окна. Она вызывает для каждой точки прямоугольника функцию, зависящую от координат точки и от положения наблюдателя и вычисляющую цвет точки

Метод трассировки лучей описан в учебном пособии
***Хусаинов А.А., Михайлова Н.Н. Программирование графики в Borland C++. –* КнАГТУ, 2009.**

Как писать параллельные программы?

- **Эта подпрограмма всегда имеет один аргумент типа (void *).**
- **Если подпрограмма имеет несколько параметров, то они передаются через структуру. В частности для метода трассировки аргумент указывает на структуру, содержащую номер окна. При числе окон=10, номер = 0, …, 9:**



Как писать параллельные программы?

- **Например:**

```
struct arg
{
  int ithr       ;       …
};
DWORD WINAPI Part(void *a)
{
  int it=((arg *)a)->ithr;  …
}
```

- **Эти подпрограммы вызываются главной программой, или подпрограммами, с помощью функции CreateThread(). Например, для метода трассировки лучей**

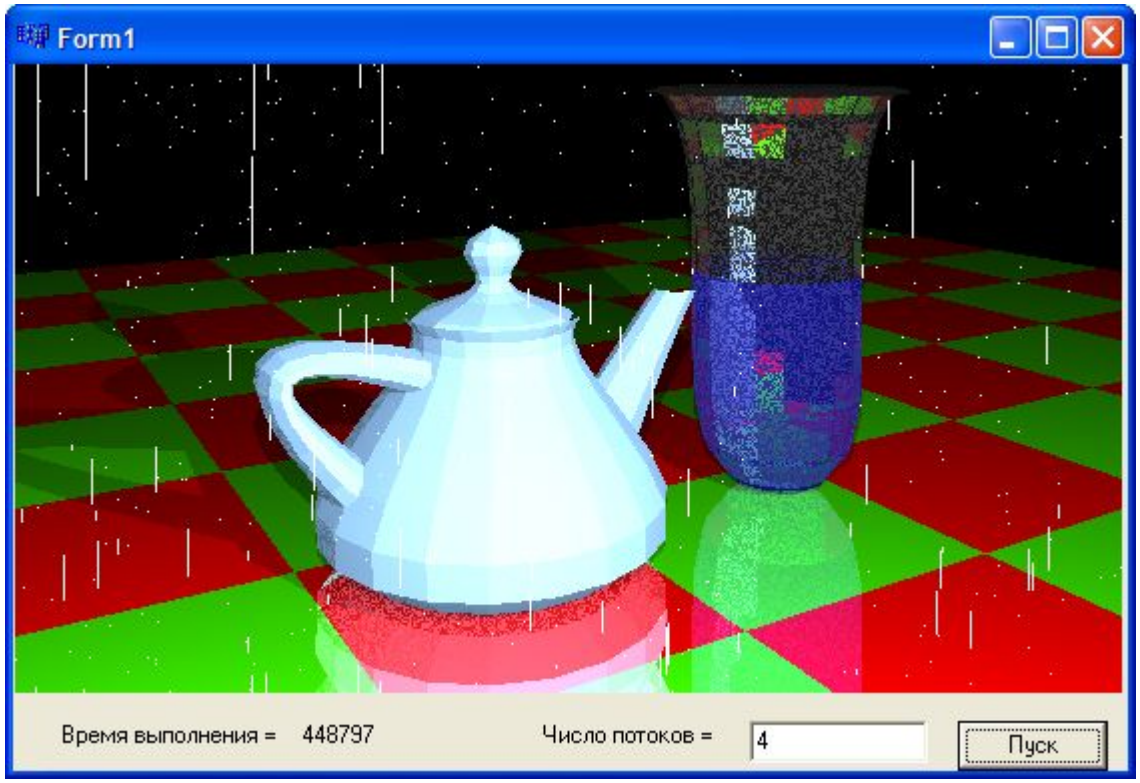
```
for (i=0;i<nthr;i++)
{
  a[i].ithr = i;  H[i]=CreateThread(NULL,0,Part,(void *)(&a[i]),0,0);
}
for (i=0;i<nthr;i++) WaitForSingleObject(H[i],INFINITE);
```

Хусаинов А.А., Михайлова Н.Н. Архитектура вычислительных систем, 2007

Итеративный параллелизм и семафоры

- Метод трассировки лучей реализуется с помощью цикла, в котором на каждом шаге вызывается некоторая подпрограмма, причем подпрограммы работают независимо.
- Это позволяет нам запускать эти подпрограммы одновременно как *потоки*.
- Поскольку число процессов намного меньше числа точек, то мы объединили части цикла в подпрограммы. И запускаем потоки параллельно.
- К сожалению, экран не позволяет различным потокам выводить точки на экран одновременно
- В частности, при числе потоков равном 4, мы получаем следующую картину:

Итеративный параллелизм и семафоры



Итеративный параллелизм и семафоры

- **Для того, чтобы исправить это, введем**

  Определение. ***Семафором Дейкстры*** **называется структура данных, состоящая из неотрицательного числа s (*счетчика семафора*) и двух унарных операций P(s) и V(s).**

  Операция V(s) увеличивает s на 1.

  Операция P(s) сначала зависает на то время, пока s = 0 . Когда будет выполнено условие s>0, P(s) отнимет от s единицу и продолжит выполнение.

- **Если имеется некоторый разделяемый ресурс, в данном случае экран, то, для того, чтобы в каждый момент времени им мог воспользоваться 1 поток, перед использованием нужно выполнить операцию P(s), а после использования – V(s). В начале работы главной программы s=1.**

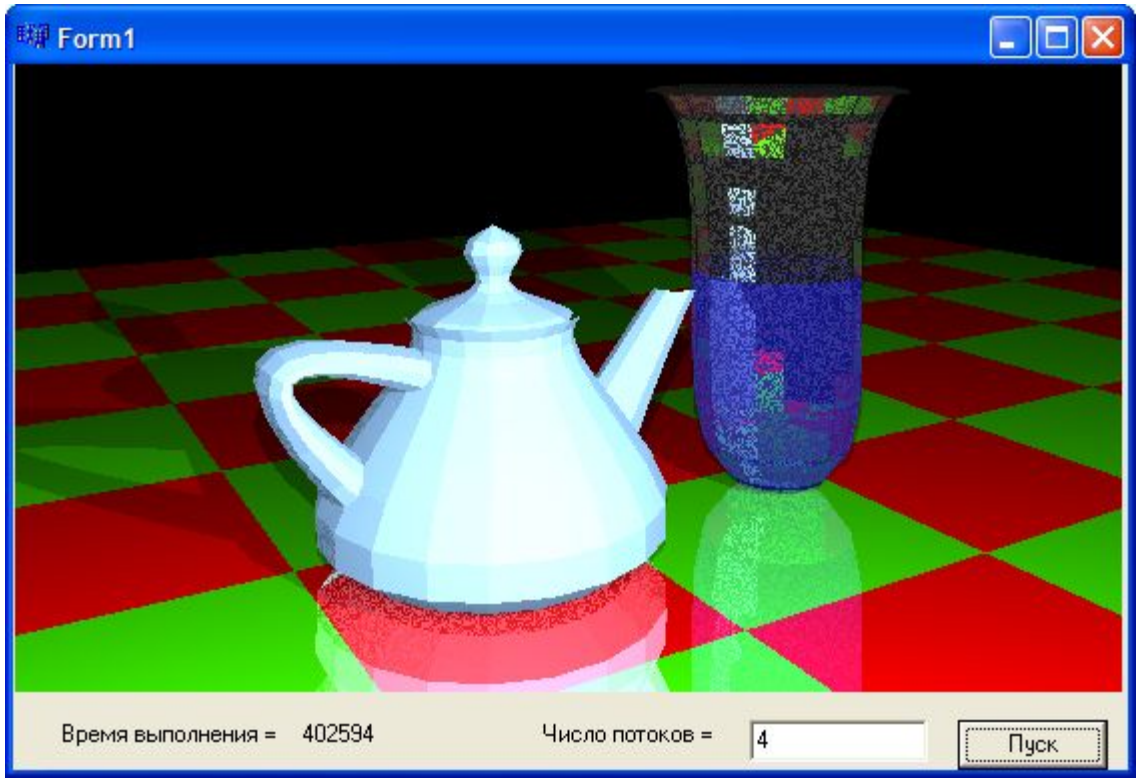
  Вывод точки осуществляется с помощью операторов

  WaitForSingleObject(sema,INFINITE); // операция P(s)
  SetPixel(dc,x,y,color); // вывод на устройство dc
  ReleaseSemaphore(sema, 1, NULL); // операция V(s)

  Предварительно семафор создается с помощью вызова

  Handle sema=CreateSemaphore(NULL,1,1,NULL);

Итеративный параллелизм и семафоры



Итеративный параллелизм и семафоры

- **Пример программы, в которой решается проблема *сериализации*. Рассмотрим задачу вычисления интеграла**

$$\int_0^1 \frac{dx}{1+x^2} \approx \frac{1}{n} \sum_{k=0}^{n-1} \left(1 / \left(1 + \left(\frac{k}{n} \right)^2 \right) \right)$$

равного площади фигуры ограниченной кривой y=1/(1+x^2) и прямыми x=0; y=0; x=1.

- **Определим данные**

```
volatile int j=0;
HANDLE mut;
double s=0;
```

- **Напишем подпрограмму добавления площади прямоугольника к сумме.**

```
DWORD WINAPI sum(void* ps)   // функция потоков
{
  int *k = (int *)ps;
  double w = (double)(1./(1.+(0. +(*k)) /n* (0. +(*k)) /n));
  WaitForSingleObject(mut, INFINITE); // ждем освобождения s
  s = s + w; j++;            // прибавляем значение
  ReleaseSemaphore(mut,1,NULL);     // освобождаем s
  return 1;
}
```

Итеративный параллелизм и семафоры

- **Напишем главную программу**

```
main()
{
  int i; int p[n];
  for(i=0; i<n; i++) p[i]=i;            // для передачи номера потока
  mut = CreateSemaphore(NULL,1,1,NULL); // создание мьютекса
  for(i=0; i<n; i++)                    // запуск потоков
  {
        CreateThread(NULL,0,sum, (void *) (&p[i]),0,0);
                                  // параметр – номер из {0, …, n-1}
  }
  while (j<n);                          // ожидание завершения всех потоков
  cout << “\nValue obtained by threads = ” << s; // результат
}
```

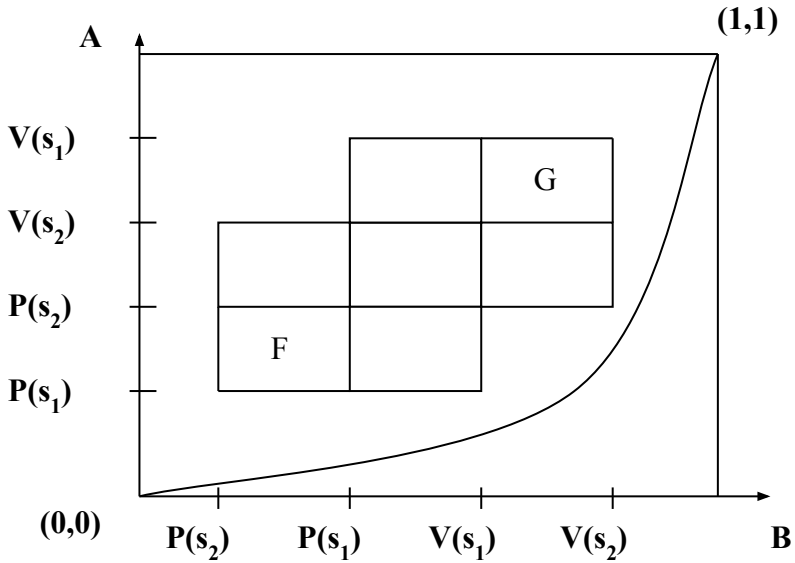
- **Результат выполнения программы будет близок к π/4.**

Математическое моделирование параллельных вычислительных систем

- При синхронизации работы потоков с помощью семафоров возникают проблемы, связанные с обнаружением тупиков и описанием пространства состояний вычислительного процесса. Рассмотрим, например, два потока, с двумя семафорами s_1 и s_2.
- Первый из них, поток A, вызывает операции $P(s_1)$; $P(s_2)$; $V(s_2)$; $V(s_1)$
- Второй, поток B, -
  $P(s_2)$; $P(s_1)$; $V(s_1)$; $V(s_2)$

Математическое моделирование параллельных вычислительных систем

- **Рассмотрим математическую модель**



Шведский флаг

состоящую из множества состояний, заданных парами точек плоскости (t_1,t_2), где t_i – доля выполнения i-го потока. Область F будет состоять из тупиков, область G – из недоступных точек. В общем случае возникают

- **Направленные топологические пространства**
- **Автоматы высших размерностей.**

Математическое моделирование параллельных вычислительных систем

- *Категория* состоит из *объектов* $A,B,C,\ \ldots$
  и *морфизмов* $A \xrightarrow{\alpha} B\ ,\ B \xrightarrow{\beta} C\ ,\ C \xrightarrow{\gamma} D\ ,\ \ldots$

- *Для любых морфизмов* $A \xrightarrow{\alpha} B$ *и* $B \xrightarrow{\beta} C$ *задана композиция* $A \xrightarrow{\beta\alpha} C$ *такая, что имеет место* $\gamma(\beta\alpha) = (\gamma\beta)\alpha$
- Для каждого объекта задан тождественный морфизм $1_A: A \to A$ такой, что $\alpha\ 1_A = \alpha$, $1_B \alpha = \alpha$

- *Функтор между категориями сопоставляет объектам - объекты, морфизмам – морфизмы, он переводит тождественные морфизмы в тождественные, композицию – в композицию.*

Математическое моделирование параллельных вычислительных систем

- Пусть U: **A** →**B** - функтор. Объект A называется *универсальным* для B ∈**B**, если задан морфизм *η: B → U(A)* такой, что для любого *η': B → U(A')* ∃! *α: A→A'*, для которого *U(α)η= η'.*
- Примеры: *Пополнение* метрического пространства является универсальным объектом. Пополнением пространства рациональных чисел будет пространство вещественных чисел. Универсальным объектом для любого множества *E* по отношению к забывающему функтору *Vect → Set* будет линейное пространство с базисом *E.*
- Как связаны между собой категории моделей вычислительных систем?

Полукубические множества

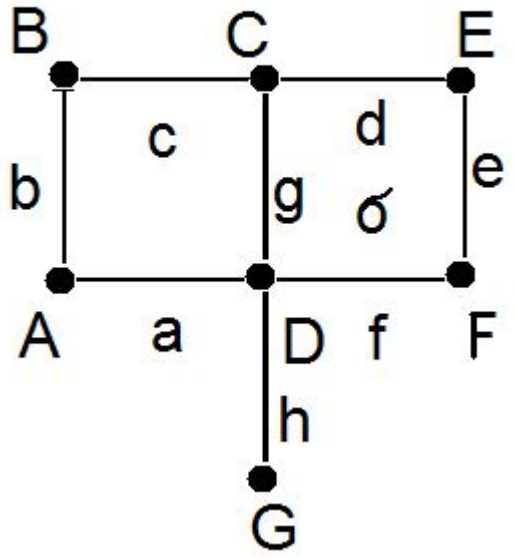
$$X = (X_n, \partial_i^{n,\varepsilon}), n = 0,1,2,\ldots; \partial_i^{n,\varepsilon} : X_n \to X_{n-1},$$

$$1 \le i \le n,\ \varepsilon \in \{0,1\}$$

$$\begin{array}{ccc} X_n & \xrightarrow{\partial_j^{n,\beta}} & X_{n-1} \\ {\scriptstyle \partial_i^{n,\alpha}}\downarrow & & \downarrow{\scriptstyle \partial_i^{n-1,\alpha}} \\ X_{n-1} & \xrightarrow[\partial_{j-1}^{n-1,\beta}]{} & X_{n-2} \end{array}$$

$$\alpha, \beta \in \{0,1\},\ n \ge 2, 1 \le i < j \le n$$

Полукубические множества



$$\partial_1^{2,0}(\sigma) = f, \partial_1^{2,1}(\sigma) = d,$$

$$\partial_2^{2,0}(\sigma) = g, \partial_2^{2,1}(\sigma) = e$$

$$X_0 = \{A, B, C, D, E, F, G\}$$

$$X_1 = \{a, b, c, d, e, f, g, h\}$$

$$X_2 = \{\sigma\}$$

$$\partial_1^{1,0}(a) = A, \partial_1^{1,1}(a) = D,$$

$$\partial_1^{1,0}(b) = A, \partial_1^{1,1}(b) = B,$$

$$\partial_1^{1,0}(c) = B, \partial_1^{1,1}(c) = C,$$

$$\partial_1^{1,0}(d) = C, \partial_1^{1,1}(d) = E,$$

$$\partial_1^{1,0}(e) = F, \partial_1^{1,1}(e) = E,$$

$$\partial_1^{1,0}(f) = D, \partial_1^{1,1}(f) = F,$$

$$\partial_1^{1,0}(g) = D, \partial_1^{1,1}(g) = C,$$

$$\partial_1^{1,0}(h) = G, \partial_1^{1,1}(h) = D$$

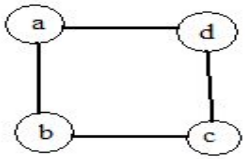
Моноиды трасс и вычислительные процессы

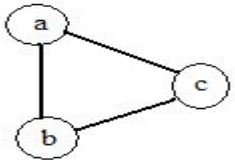
Что такое вычислительный процесс?

Рассмотрим вычислительную систему, состоящую из операций

Определение. Пусть E – мн-во, I ⊆ExE – наз *отношением независимости*, если I антирефлексивно и симметрично

M(E,I)=⟨E | ∀(a,b) ∈I (ab=ba)⟩ - *свободный частично-коммутативный моноид* или *моноид трасс*





Граф независимости: вершины E, ребра {{a,b}:(a,b) ∈I}

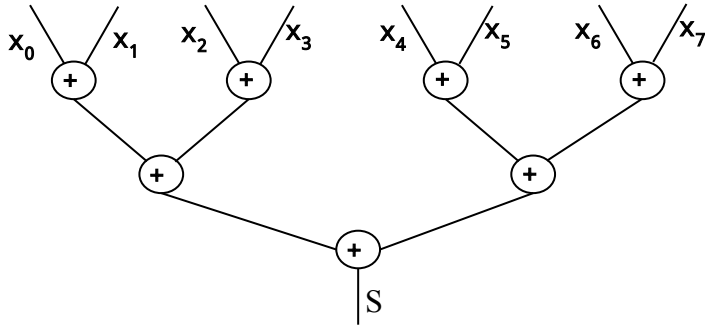
Операции вычислительной системы порождают свободный частично коммутативный моноид. *Вычислительный процесс* – композиция этих операций. *Язык Мазуркевича* состоит из независающих вычислительных процессов.

Моноиды трасс и полукубические множества

- Теорема 1. Каждому полукубическому множеству с выделенной вершиной соответствует универсальный моноид трасс
- Соответствующий моноид будет порожден ребрами - 1-кубиками. Перестановочны соседние ребра из 2-кубиков. Отождествляются противоположные.
- Эта теорема позволяет определить сохраняющие независимость морфизмы полукубических множеств.

Рекурсивный параллелизм

- **Метод сдваивания**



Вычисление суммы методом сдваивания s=sum(0,n-1);

```
int sum(int l, int r) // x[l] + … + x[r]
{
      if(l == r) return x[l];
      else return sum(l, (l + r + 1)/2 - 1) + sum((l + r + 1)/2, r);
}
```

Рекурсивный параллелизм

- **Данные и структура параметров**
  int x[100]; struct arg {int l, r, res};
- **Вызываемый поток**

```
DWORD WINAPI sum(void* s)
{
  int i, l=((arg *)s)->l, r=((arg *)s)->r;
  if (l==r) ((arg *)s)->rez = x[l];        // результат
        else
        {// в противном случае вызываем два потока с разными аргументами
        arg *r1 = new arg, *r2 = new arg;
        HANDLE H[2];
        r1->l=l; r1->r= (l+r+1)/2-1; r2->l=(l+r+1)/2; r2->r = r;                    H[0]=
CreateThread(0,0,sum, (void *)r1,0,0);
        H[1]= CreateThread(0,0,sum, (void *)r2,0,0);
        WaitForMultipleObjects(2, H, 1, INFINITE);
        ((arg *)s)->rez = (r1->rez)+(r2->rez);
        delete r1; delete r2;
} return 1;
}
```

- **Вызов из главной программы**
  arg t; t.l = 0; t.r = 99;   sum(&t);

Рекурсивный параллелизм

- Рекурсивный параллелизм применяется для распараллеливания алгоритма перебора с возвратом
- И.А. Трещев установил в программе счетчик, для определения, нужно ли загружать поток, если существуют свободные процессоры, или вызывать этот модуль как рекурсивную подпрограмму. Оказалось, что наибольшее ускорение достигается при числе потоков, большем, чем число процессоров.

Конвейерные системы

- Рассмотрим вычислительную систему для вычисления суммы векторов. Она состоит из 5 микроопераций
- Comp(a,b) – сравнение порядков
- Shift(a,b) – сдвиг мантиссы большего числа
- Add(a,b) – сложение мантисс
- Norm(a,b) – нормализация
- Round(a,b) – округление результата

Для сложения двух векторов (a_1, …,a_n) и (b_1, …,b_n) в 1-й такт выполняется Comp(a_1,b_1), во 2-й Comp(a_2,b_2) Shift(a_1,b_1) и т.д. :

Конвейерные системы

	Comp	Shift	Add	Norm	Round
h	(a_1,b_1)				
2h	(a_2,b_2)	(a_1,b_1)			
3h	(a_3,b_3)	(a_2,b_2)	(a_1,b_1)		
4h	(a_4,b_4)	(a_3,b_3)	(a_2,b_2)	(a_1,b_1)	
5h	(a_5,b_5)	(a_4,b_4)	(a_3,b_3)	(a_2,b_2)	(a_1,b_1)
…	…	…	…	…	…
nh	(a_n,b_n)	(a_{n-1},b_{n-1})	(a_{n-2},b_{n-2})	(a_{n-3},b_{n-3})	(a_{n-4},b_{n-4})
(n+1)h		(a_n,b_n)	(a_{n-1},b_{n-1})	(a_{n-2},b_{n-2})	(a_{n-3},b_{n-3})
(n+2)h			(a_n,b_n)	(a_{n-1},b_{n-1})	(a_{n-2},b_{n-2})
(n+3)h				(a_n,b_n)	(a_{n-1},b_{n-1})
(n+4)h					(a_n,b_n)

Ускорение $S_5=T_1/T_5=5nh/((n+4)h) \approx 5$

Производители и потребители: каналы

```
template<class Type>
class channel
{
   Type *buf; // буфер для очереди
   int size;  HANDLE s, // семафор доступа к очереди
   empty, full; // число свободных и заполненных мест в очереди
   int countr, countw; // счетчики чтения и записи
public:
   channel (int n): size(n) // constructor
   {
            buf = new Type [n];
            countr=0; countw=0;
            s=CreateSemaphore(NULL,1,1,NULL); //
            empty=CreateSemaphore(NULL,n,n,NULL); // n свободных мест
            full=CreateSemaphore(NULL,0,n,NULL); // no full places
   }
   void operator << (Type d) // запись в канал
   {
            WaitForSingleObject(empty, INFINITE); // ждем своб мест
            WaitForSingleObject(s, INFINITE); // захват буфера
            buf[countw++]=d; // запись в очередь
            if (countw==size) countw=0;
            ReleaseSemaphore(s,1,NULL);
            ReleaseSemaphore(full,1,NULL); // full++
   }
   void operator >> (Type& d) // чтение из канала
   {
            WaitForSingleObject(full, INFINITE); // ждем поступления данных
            WaitForSingleObject(s, INFINITE); // захват буфера
            d = buf[countr]; countr++; // чтение из очереди
            if (countw==size) countw=0;
            ReleaseSemaphore(s,1,NULL);
            ReleaseSemaphore(empty,1,NULL); // empty++
   }
};
```

Производители и потребители: каналы

- С помощью каналов можно реализовать конвейерные системы.
- Блок конвейера

```
DWORD WINAPI name_op(void *)
{
  Type d;
  while(1)
  {
        ch[in]>>d; ch[out]<<op(d);
  }
}
```

Класс channel был разработан в препринте

Husainov A.A. The study of distributed computing algorithms by multithread applications // Preprint, Cornell Univ. 2004. 17 pp. **http://arxiv.org/abs/cs.DC/0404015**

Клиент-сервер: задача о читателях и писателях

- Процессы читают и редактируют файл
- Имеющие право на чтение – читатели
- На чтение-запись – писатели
- Писатель может работать только один
- Читателей может быть несколько
- Писатели имеют приоритет перед читателями – если писатель поступил, то читатели не могут работать
- Поступивший писатель становится в очередь

Элементы параллельного программирования / под ред. В.Е. Котова – М.: Радио и связь, 1983

Асинхронные системы

$T=(S,s_0,E,I,Tran)$

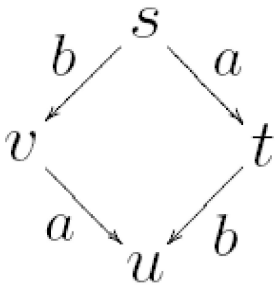
S – множество *состояний*,

$s_0 \in S$ – *начальное состояние*,

E – множество *событий*,

$Tran \subseteq S \times E \times S$ – множество *переходов*

$I \subset E \times E$ – антирефлексивное симметричное отношение *независимости*



(i) $(\forall\, a \in E\ \exists\, s, s' \in S)\ (s,a,s') \in Tran$

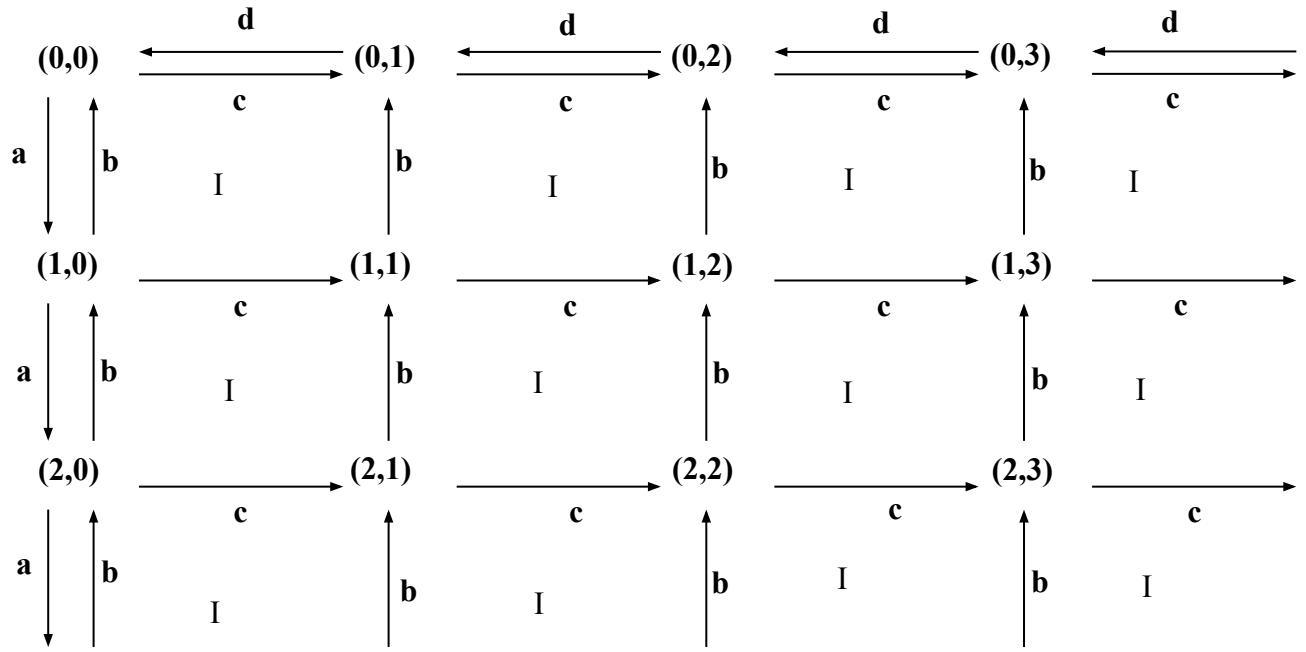
(ii) $(s,a,s') \in Tran \ \&\ (s,a,s'') \in Tran \Rightarrow s' = s''$

(iii) $(a,b) \in I \ \&\ (s,a,t) \in Tran \ \&\ (t,b,u) \in Tran \Rightarrow$
$(\exists\, v \in S)\ (s,b,v) \in Tran \ \&\ (v,a,u) \in Tran$

Пунктированное множество с действием свободного частично коммутативного моноида

Асинхронные системы

a – читатель поступил доступ
b – читатель закончил работу с файлом
c – поступил новый писатель
d – писатель закончил работу с файлом



Асинхронные системы и кубические множества

Теорема 2. Каждой асинхронной системе соответствует некоторое кубическое множество. И наоборот, каждому пунктированному кубическому множеству соответствует универсальная асинхронная система.

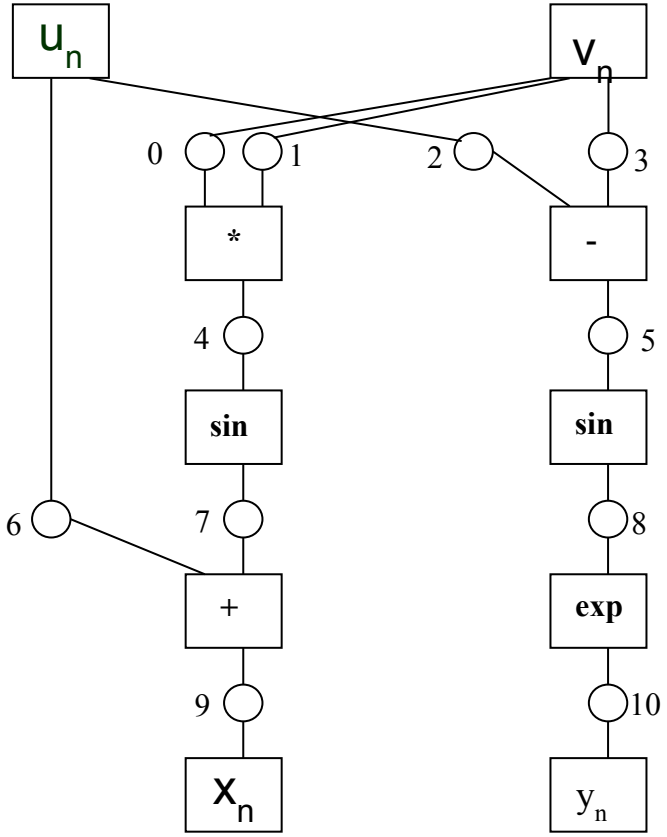
Асинхронные системы и кубические множества

- Асинхронным системам соответствуют также полиэдры – топологические пространства, склеенные из точек, отрезков, треугольников и т.д.
- Топологическим свойствам асинхронных систем посвящена статья

Хусаинов А.А., Лопаткин В.Е., Трещев И.А. Исследование математической модели параллельных вычислительных систем методами алгебраической топологии // Сиб.ЖИМ №1, с. 141-151 (2008)

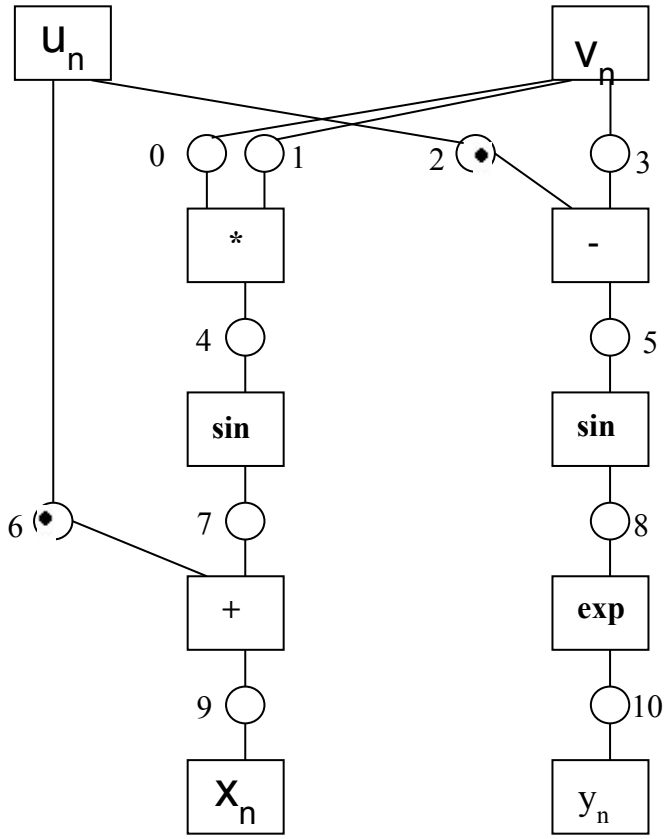
Взаимодействующие каналы

Сеть Петри волновой системы: $x_n=u_n+\sin(v_n*v_n)$, $y_n=\exp(\sin(u_n-v_n))$



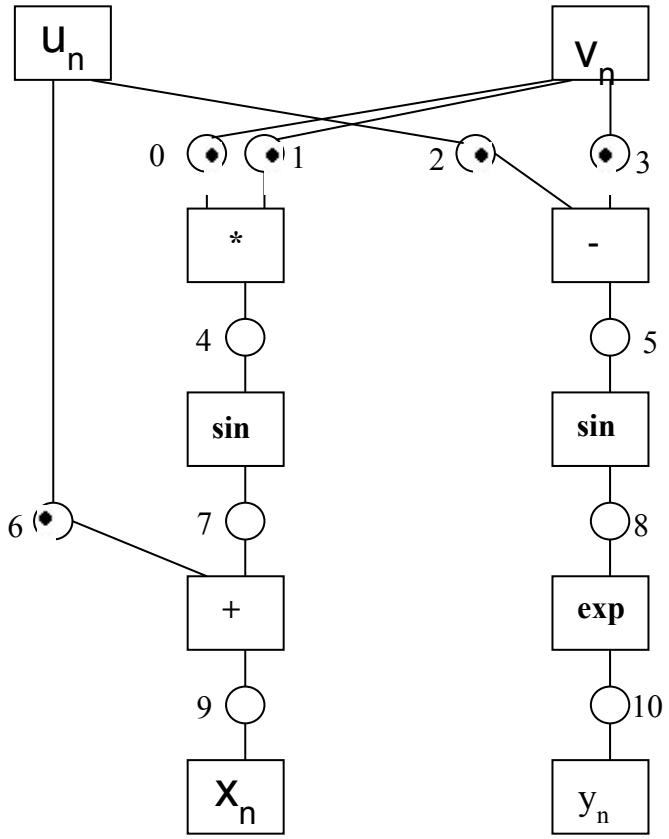
Взаимодействующие каналы

Сеть Петри волновой системы: $x_n=u_n+\sin(v_n*v_n)$, $y_n=\exp(\sin(u_n-v_n))$



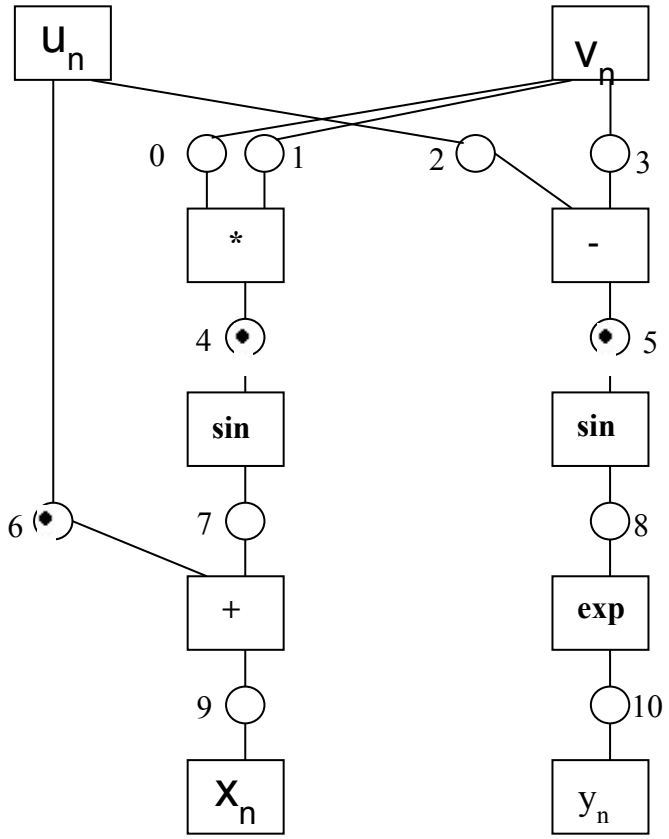
Взаимодействующие каналы

Сеть Петри волновой системы: $x_n=u_n+\sin(v_n*v_n)$, $y_n=\exp(\sin(u_n-v_n))$



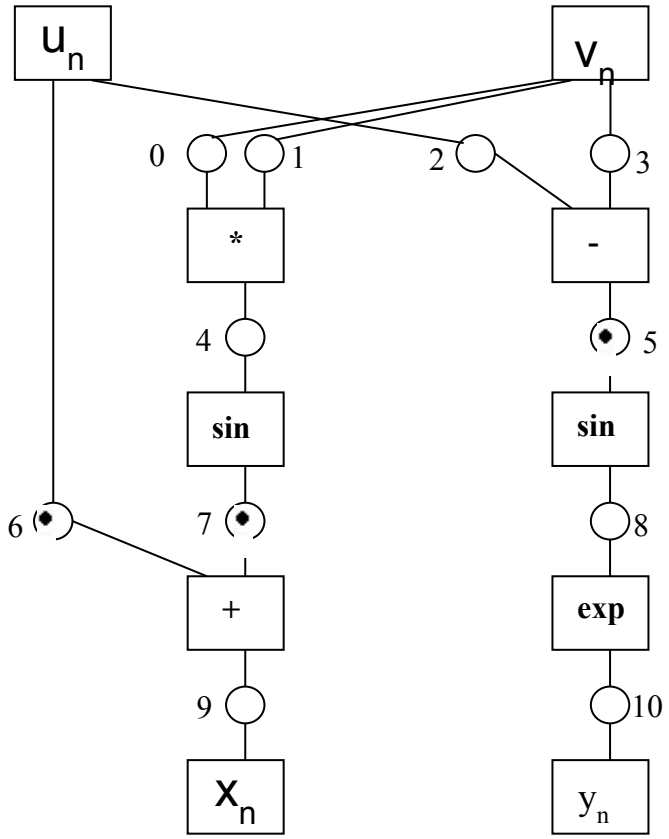
Взаимодействующие каналы

Сеть Петри волновой системы: $x_n=u_n+\sin(v_n*v_n)$, $y_n=\exp(\sin(u_n-v_n))$



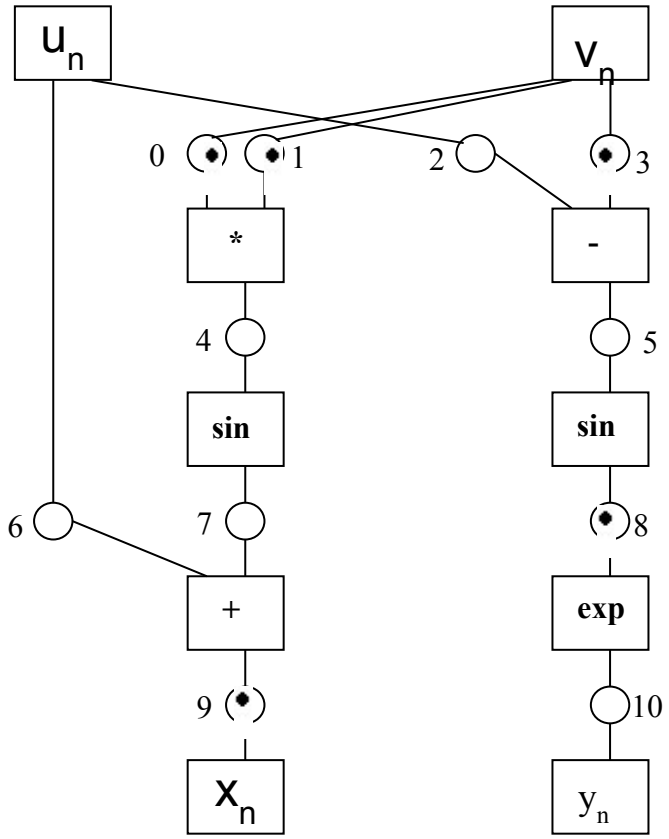
Взаимодействующие каналы

Сеть Петри волновой системы: $x_n=u_n+\sin(v_n*v_n)$, $y_n=\exp(\sin(u_n-v_n))$



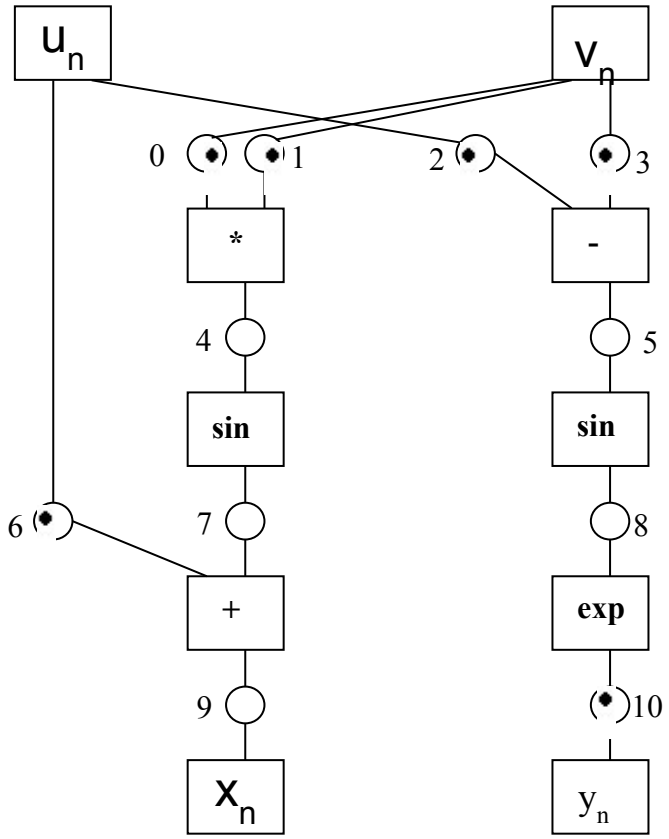
Взаимодействующие каналы

Сеть Петри волновой системы: $x_n=u_n+\sin(v_n*v_n)$, $y_n=\exp(\sin(u_n-v_n))$



Взаимодействующие каналы

Сеть Петри волновой системы: $x_n=u_n+\sin(v_n*v_n)$, $y_n=\exp(\sin(u_n-v_n))$



Взаимодействующие каналы

- Каналы, соответствующие местам
  channel *pc[11];
- Подпрограмма потока

```
DWORD WINAPI mult(LPVOID)        // поток для умножения
{
  int j;
  double d1, d2;
  for (j=0; j<n; j++)            // цикл
  {
    *pc[0]>>d1; *pc[1]>>d2; // прием данных из каналов 0 и 1
    *pc[4]<<(d1*d2);      // умножение и отправление в канал 4
  }
  return 1;
}
```

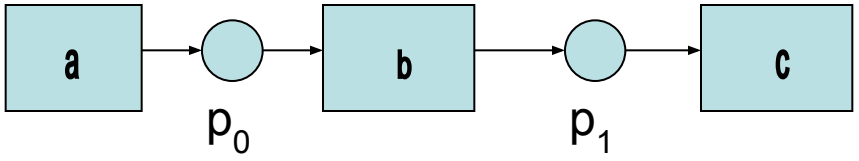
Взаимодействующие каналы

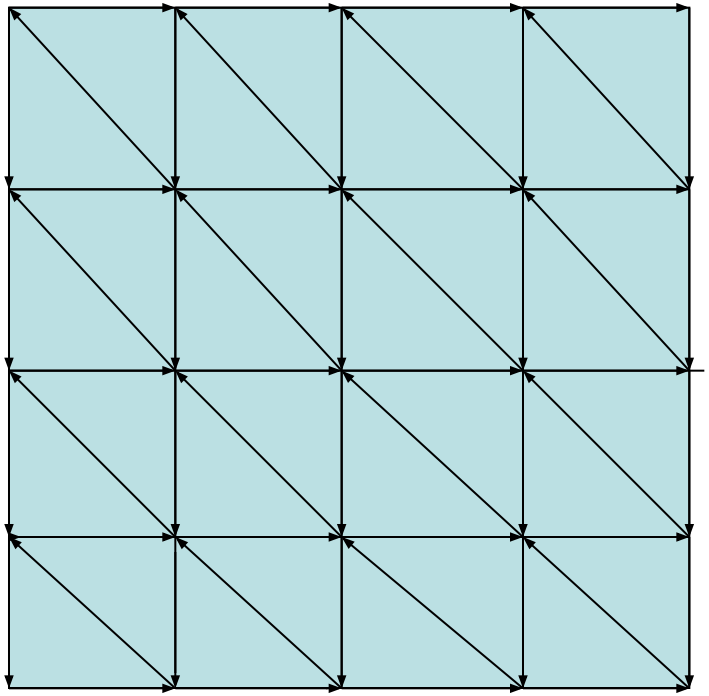
- Полученные «асинхронные систолические системы» называются волновыми системами
- Более точная математическая модель волновой системы, чем сеть Петри, была разработана и применена для оценки ускорения в диссертации

И.А. Трещев «МАТЕМАТИЧЕСКИЕ МОДЕЛИ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ И ИХ ПРИМЕНЕНИЕ ДЛЯ ПОСТРОЕНИЯ МНОГОПОТОЧНЫХ ПРИЛОЖЕНИЙ НА СИСТЕМАХ С SMP-АРХИТЕКТУРОЙ »

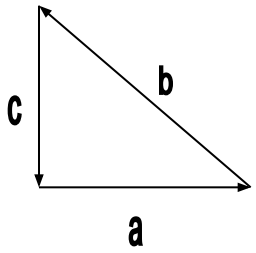
Сети Петри и асинхронные системы

Пример





$M(E,I)= \langle a,b,c | ac=ca \rangle$

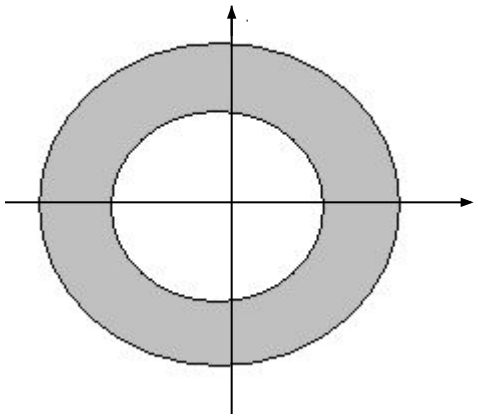


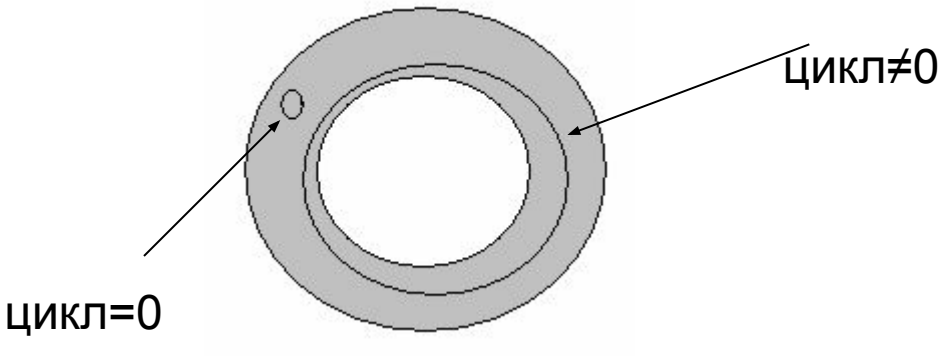
Топология – «резиновая» геометрия (изучает инварианты гомеоморфизмов)

Числа Бетти

Кольцо

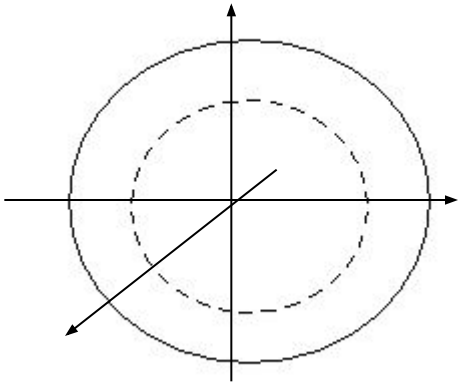
$\{(x,y): r^2 \le x^2+y^2 \le R^2\}$





Каждой дырке соответствует цикл, не являющийся границей подобласти. В данном случае существует 1-мерный цикл
не ограничивающий подобласть

Числа Бетти.



$\beta_1 = 0, \ \beta_2 = 1$

Резиновый мяч. Любой 1-мерный цикл является границей поверхности, содержащейся в области

$$\{(x,y,z): r^2 \le x^2+y^2+z^2 \le R^2\} .$$

В данном случае существует 2-мерный цикл, окружающий дырку, и поэтому не являющийся границей.

Числа Бетти

Разбиваем фигуру на треугольники, тетраэдры, …, n-симплексы (x_0, …, x_n)
Цепи – суммы n-симплексов

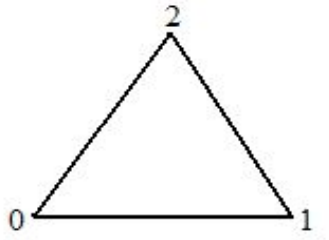
$C_n = L\{(x_0, x_1, \ldots, x_n) –$ *симплекс:* $x_0 < x_1 < \ldots < x_n\}$
$Z_n = d_n^{-1}(0)$, $B_n = d_{n+1} C_{n+1}$
$H_n = Z_n / B_n$
$\beta_n = \dim C_n - r(d_n) - r(d_{n+1})$

.

$$0 \leftarrow C_0 \xleftarrow{d_1} C_1 \xleftarrow{d_2} \ldots \xleftarrow{d_n} C_n \xleftarrow{d_{n+1}} \ldots$$

$$d_n(x_0, x_1, \ldots, x_n) = \sum_{i=0}^{n} (-1)^i (x_0, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$$

Числа Бетти



$$0 \leftarrow C_0 \xleftarrow{d_1} C_1 \leftarrow 0$$

2-цепей нет ⇒
цикл 01+12-02 ≠ 0

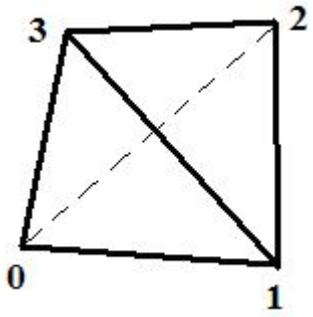
симплексы	01	02	12
0	-1	-1	0
1	1	0	-1
2	0	1	1

$\beta_0=3-0-r(d_1)=3-2=1$, $\beta_1=3-r(d_1)-0=1$

Вычисление чисел Бетти

$$0 \leftarrow C_0 \xleftarrow{d_1} C_1 \xleftarrow{d_2} \cdots \xleftarrow{d_n} C_n \xleftarrow{d_{n+1}} \cdots$$

$$\beta_n = \dim C_n - r(d_n) - r(d_{n+1})$$

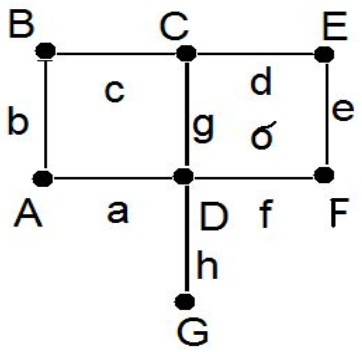


$C_0 = L\{0,1,2,3\} \approx \boldsymbol{Q}^4$,
$C_1 = L\{01,02,03,12,13,23\} \approx \boldsymbol{Q}^6$,
$C_2 = L\{012,013,023,123\} \approx \boldsymbol{Q}^4$
rank d_1=3, rank d_2=3 ⇒ β_0=1, β_1= 0, β_2=1.

	01	*02*	*03*	*12*	*13*	*23*
0	-1	-1	-1	0	0	0
1	1	0	0	-1	-1	0
2	0	1	0	1	0	-1
3	0	0	1	0	1	1

	012	*013*	*023*	*123*
01	1	1	0	0
02	-1	0	1	0
03	0	-1	-1	0
12	1	0	0	1
13	0	1	0	-1
23	0	0	1	1

Числа Бетти полукубических множеств



d_1=

	a	b	c	d	e	f	g	h
A	1	1						
B		-1	1					
C			-1	1			-1	
D	-1					1	1	-1
E				-1	-1			
F					1	-1		
G								1

d_2=

	σ
a	
b	
c	
d	-1
e	1
f	1
g	-1
h	

$$d_n = \sum_{i=1}^{n} (-1)^i (L(\partial_i^{n,1}) - L(\partial_i^{n,0}))$$

$$0 \leftarrow Q^7 \xleftarrow[d_1]{} Q^8 \xleftarrow[d_2]{} Q \leftarrow 0$$

$$rk(d_1) = 6,\ rk(d_2) = 1 \quad \Rightarrow$$

$$\beta_0 = 1, \beta_1 = 8 - 6 - 1 = 1, \beta_2 = 1 - 1 = 0$$

Числа Бетти асинхронных систем

$S_0 = S \cup \{*\}$, $S_1 = \{(s, e_1): s \in S_0, e_1 \in E\}$, … ,

$S_n = \{(s, e_1, \ldots, e_n): s \in S_0, e_1 < \ldots < e_n \in E, (e_i, e_j) \in I, \text{ при } 1 \le i < j \le n\}$

Предложение. Числа Бетти асинхронной системы вычисляются с помощью комплекса

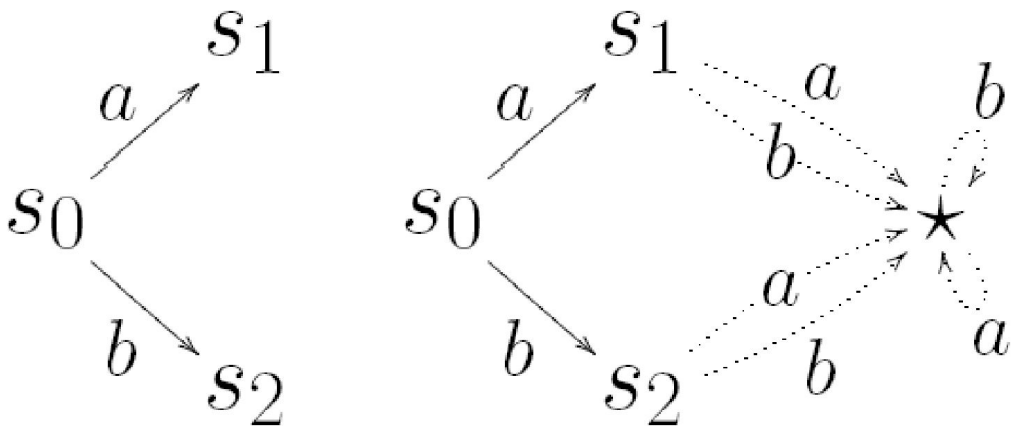
$$0 \leftarrow LS_0 \xleftarrow{d_1} LS_1 \xleftarrow{d_2} \ldots \xleftarrow{d_n} LS_n \xleftarrow{d_{n+1}} \ldots$$

$$d_n(s, e_1, \ldots, e_n) = \sum_{i=1}^{n} (-1)^i ((se_i, e_1, \ldots, \hat{e}_i, \ldots, e_n) - (s, e_1, \ldots, \hat{e}_i, \ldots, e_n))$$

В частности $d_1(s, e_1) = -se_1 + s$

$$d_2(s, e_1, e_2) = -(se_1, e_2) + (s, e_2) + (se_2, e_1) - (s, e_1)$$

Числа Бетти асинхронных систем



$$0 \leftarrow Q^4 \xleftarrow{d_1} Q^8 \xleftarrow{d_2} Q^4 \leftarrow 0$$

	s_0 a b	s_1 a b	s_2 a b	*a b
(s_0,a)	-1			
(s_0,b)	1			
(s_1,a)	1	-1		
(s_1,b)	-1	1		
(s_2,a)			-1	
(s_2,b)			1	
(* ,a)		1	1	0
(* ,b)		-1	-1	0

	s_0,a	s_0,b	s_1,a	s_1,b	s_2,a	s_2,b	*a	*b
s_0	1	1						
s_1	-1		1	1				
s_2		-1			1	1		
*			-1	-1	-1	-1	0	0

rk d_1 = 3 *rk d_2 = 3*

β_0=4-3=1 , β_1=8-3-3=2 , β_2=4-3=1

Числа Бетти асинхронных систем

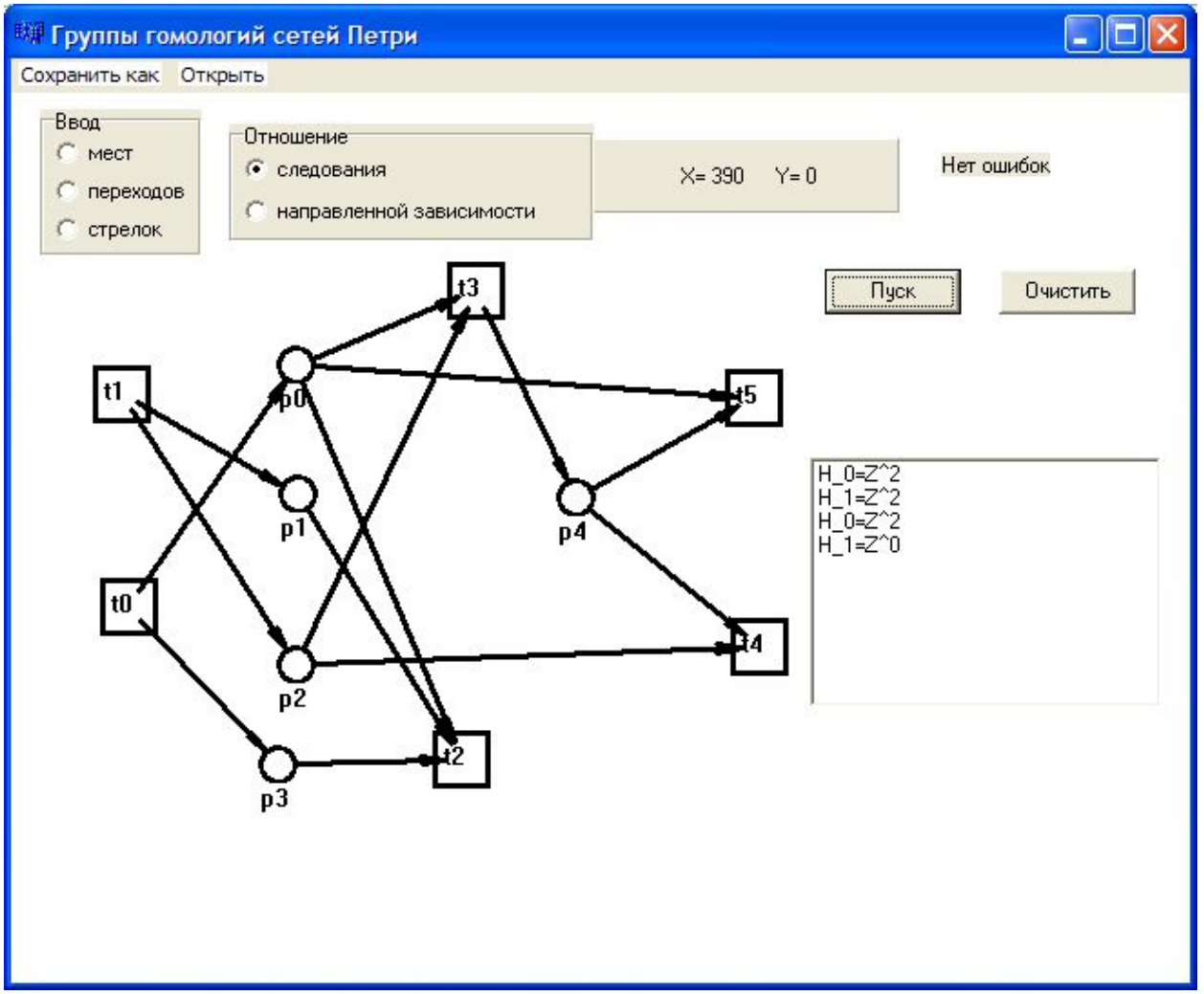
- 28 октября (четверг), в 12.00, в 201/3 состоится защита диссертации В.Е. Лопаткина «Исследование математических моделей параллельных вычислительных систем методами алгебраической топологии», в ней развиваются
- Приложения чисел Бетти для нахождения признаков распараллеливаемости
- Другие инварианты: группы гомологий асинхронных систем, кольца когомологий автоматов высших размерностей и их свойства

Числа Бетти сетей Петри

2 подхода к определению чисел Бетти сетей Петри:

- Сети Петри сопоставляется асинхронная система и вычисляются ее числа Бетти (***Husainov A.A. Homology of small categories and asynchronous transition systems // Homology, Homotopy and Applications 2004***). В качестве примера, вычислены числа Бетти конвейера из трех переходов
- Для сети Петри и отношения независимости между переходами строятся 2 частично-упорядоченных множества и вычисляются числа Бетти этих множеств (А.Д. Гринблат, Е.А. Хусаинова)

Числа Бетти сетей Петри



Открытые проблемы

- Разработать метод построения многопоточного приложения по асинхронной системе
- Доказать, что первая группа гомологий асинхронной системы не имеет кручения
- Могут ли иметь кручение группы гомологий асинхронной системы элементарной сети Петри?
- Найти признаки разложимости асинхронной системы переходов в параллельную композицию взаимодействующих вычислительных процессов