Суперкомпьютерные дни в России 2019

Аспектно-ориентированный язык Set@l для архитектурно-независимого программирования высокопроизводительных вычислительных систем

Левин И.И., Дордопуло А.И., Писаренко И.В., Мельников А.К.

Научно-исследовательский центр супер-ЭВМ и нейрокомпьютеров

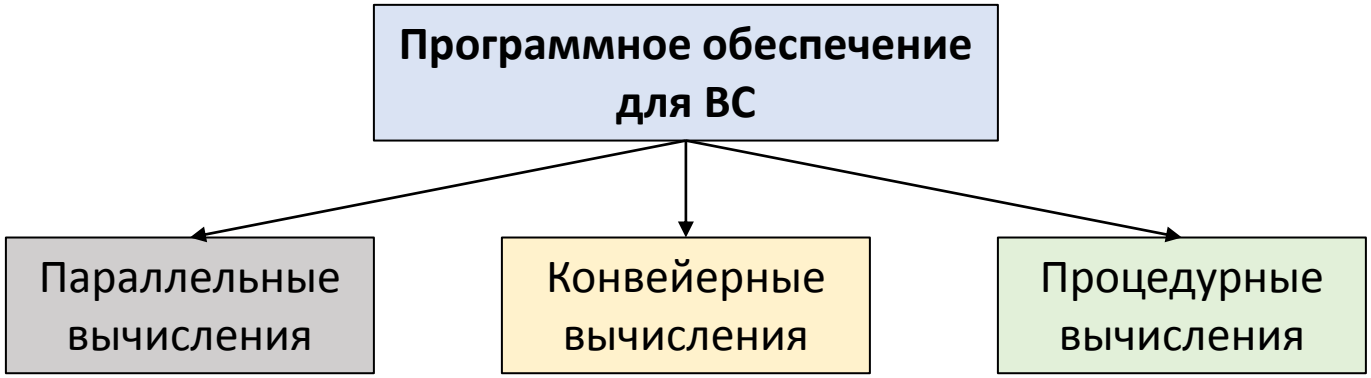
г. Таганрог, Россия

Международная конференция, 23-24 сентября 2019 г., Москва, Россия

Проблемы программирования параллельных ВС

Современные высокопроизводительные ВС характеризуются **многообразием** используемых **вычислительных архитектур**.

Одно из актуальных направление развития – создание **гибридных** (гетерогенных) ВС, объединяющих **процессоры** и **ПЛИС**.



Ключевая проблема – **портирование** ПО между разными вычислительными архитектурами.

Существующие средства программирования высокопроизводительных ВС являются **архитектурно-специализированными**: портирование программы на другую архитектуру требует разработки нового кода на другом языке программирования.

Проблемы программирования параллельных ВС

Архитектурная специализация средств программирования ВС:

OpenMP → МП ВС с общей памятью

MPI → МП ВС с распределенной памятью

OpenACC, **CUDA C**, **C++ AMP**, **ATI Stream** → CPU+GPU

COLAMO → Реконфигурируемые ВС с полем ПЛИС

OpenCL → CPU+GPU, CPU+FPGA

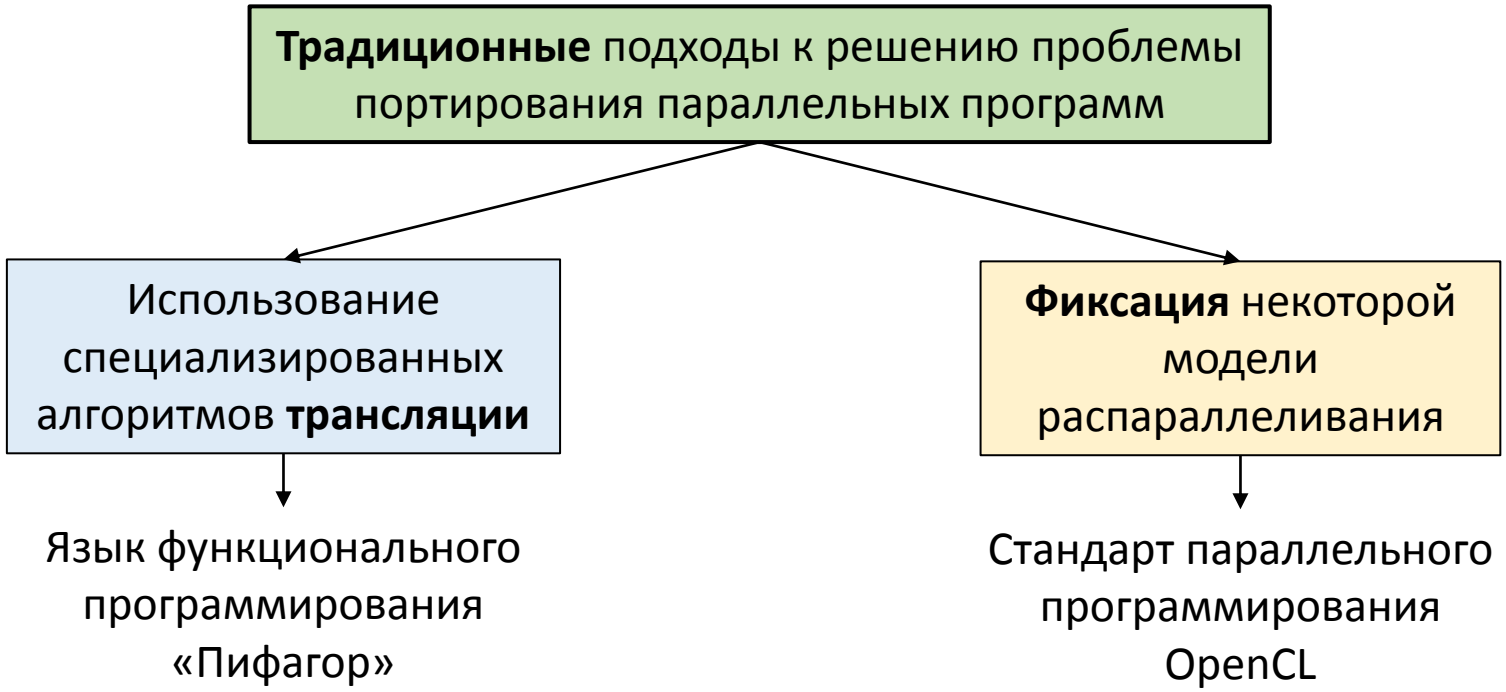
Портирование параллельных приложений:

- математическая сущность алгоритма остается неизменной;
- изменяются особенности реализации (распараллеливание, память, …).

Архитектурные ограничения существующих языков параллельного программирования обусловлены отсутствием средств, позволяющих описывать алгоритм **отдельно** от деталей его реализации на ВС.

Проблемы программирования параллельных ВС

Разработанные ранее методы архитектурно-независимого параллельного программирования характеризуются существенными **недостатками**.



Многообразие архитектур, используемых в современных ВС, и **отсутствие** эффективных методов и средств **архитектурно-независимого** параллельного программирования обуславливают актуальность разработки **новых подходов** к решению проблемы портирования прикладного ПО.

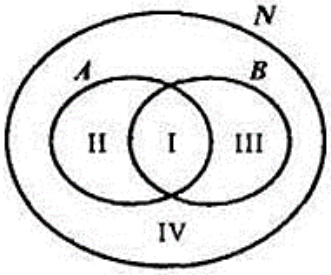
Язык архитектурно-независимого программирования ВС Set@l

Set@l (Set Aspect-Oriented Language) – язык архитектурно-независимого программирования ВС, дальнейшее развитие идей языков программирования **COLAMO** и **SETL**.

Set@l

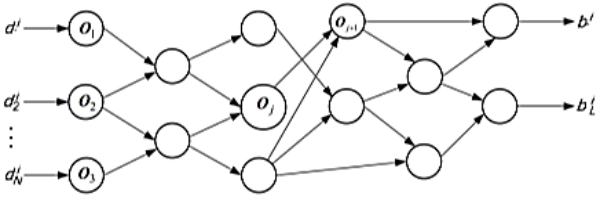
Теоретико-множественное представление кода, **реляционное** исчисление, механизмы **разбиения** совокупностей

SETL (SET Language, Jacob T. Schwartz, 1960-е гг.), SETL2, SETLX



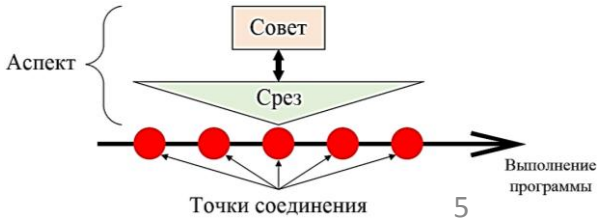
Описание **информационного графа** решаемой прикладной задачи

COLAMO



Парадигма **аспектно-ориентированного программирования** (АОП)

Gregor Kiczales, 1997 г.



Set@l и SETL

Недостатки языка SETL и других классических теоретико-множественных языков программирования:

- не предназначены для описания **параллельных вычислений** и, в сущности, являются **процедурными** языками программирования;
- оперируют только **четко заданными** совокупностями – множествами;
- не обладают **гибкостью** объектно- и аспектно-ориентированного подхода к программированию.

В отличие от языка SETL, в языке программирования Set@l используется **классификация** совокупностей **по различным критериям** – параллелизму их элементов при обработке, четкости, упорядоченности и т.д.

```
Z:={-10..10};  $ no infinite sets in SETL!
E:={n: n in Z | n mod 2 = 0};
print(E);

{0 2 4 6 8 10 -2 -4 -6 -8 -10}
```

```
N:={1..50};
P:={x: x in N | forall y in {2..x-1} | x mod y /=0};
print(P);

{1 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47}
```

Set@l и COLAMO

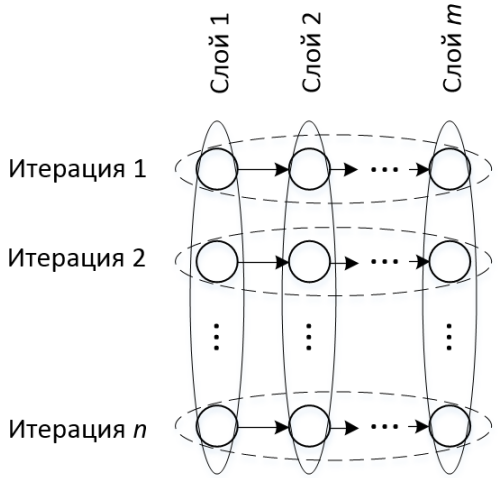
В COLAMO распараллеливание описывается в **неявной форме** путем объявления **типов доступа** к массивам (`Vector` и `Stream`) и **индексации** их элементов.

Язык COLAMO ориентирован на **структурно-процедурную** организацию вычислений, что **не позволяет** переносить описанные на нем параллельные программы **между** ВС с **различными архитектурами**.

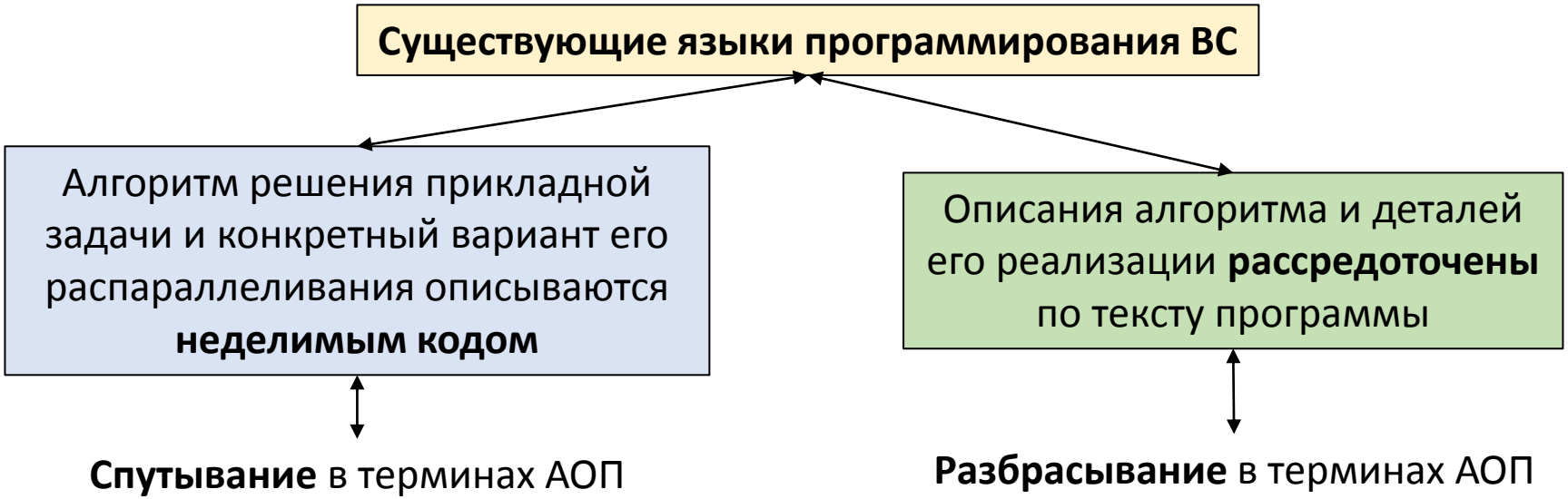
Представление **информационного графа** решаемой задачи:

- **COLAMO**: операционные вершины и массивы данных с различными типами доступа и хранения; разбиение на слои и итерации;
- **Set@l**: множества, их признаки и отношения; возможны любые типы разбиений.

```
Var a,b,c,d : Array Integer [10 : Stream] Mem;
Var i: Number;
Cadr ExampleMem;
  For i := 1 To 5 Do
    Begin
      a[i] := b[i] · c[i] + d[i];
    End;
Endcadr;
```



Аспектно-ориентированный подход к программированию ВС



Сквозная функциональность программы описывается в виде отдельных программных модулей **– аспектов**.

Исходный код содержит **разметку**, которая определяет его взаимодействие с аспектами при трансляции или исполнении.

Анализируя созданную пользователем разметку, **транслятор-препроцессор** формирует новую исполняемую программу, в которой сквозная функциональность **вплетена** в код.

Аспектно-ориентированный подход к программированию ВС

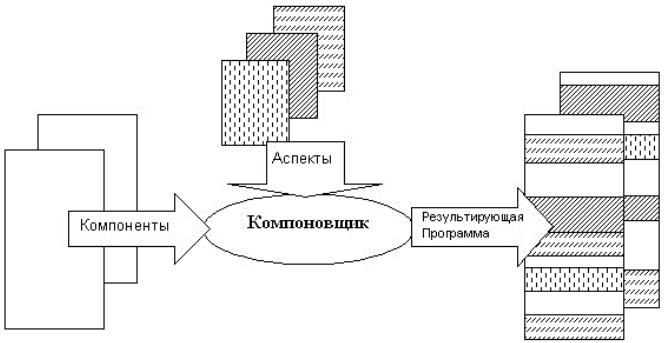
Использование технологии АОП позволяет:

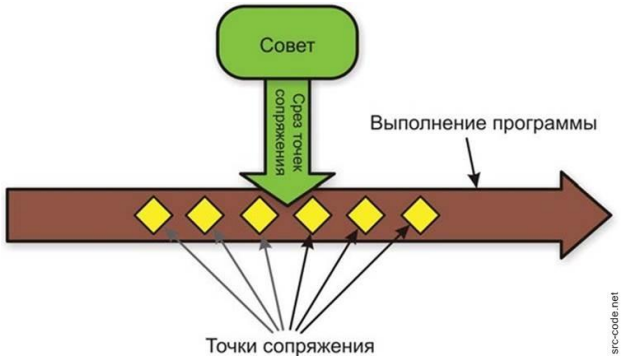
- **упростить** разработку и дальнейшую поддержку ПО;
- повысить **адаптируемость** программ к различным изменениям.

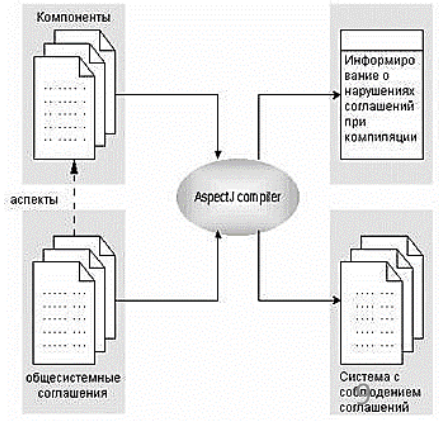
Существующие языки АОП (Aspect J, Spring AOP, Aspect C++ и др.):

- являются **расширениями** процедурных языков (Java, C++ и др.);
- направлены на дальнейшее развитие **объектно-ориентированного** подхода, а не на описание **вычислительных** задач.

Для решения проблемы портирования необходим **принципиально новый язык** программирования, который обеспечит аспектно-ориентированную декомпозицию алгоритмов и выделение особенностей их реализации на различных архитектурах ВС в отдельные программные модули.







Исходный код и аспекты программы на языке Set@l

Программа на языке Set@l состоит из следующих модулей:

- архитектурно-независимого **исходного кода** (`program`) алгоритма решения задачи;
- системы **аспектов** (`aspect`), адаптирующих алгоритм к архитектуре и конфигурации определенной ВС.

Интерфейсный раздел (`interface`) включает:

- входные параметры (`input`);
- выходные параметры (`output`);
- неопределенные параметры (`extern`);
- ссылки на аспекты-источники и аспекты-приемники.

Исходный код

```
program(cmp1):
   c = a + b;
end(cmp1);
```

Аспект

```
aspect(asp1):
   type(a)='par';
end(asp1);
```

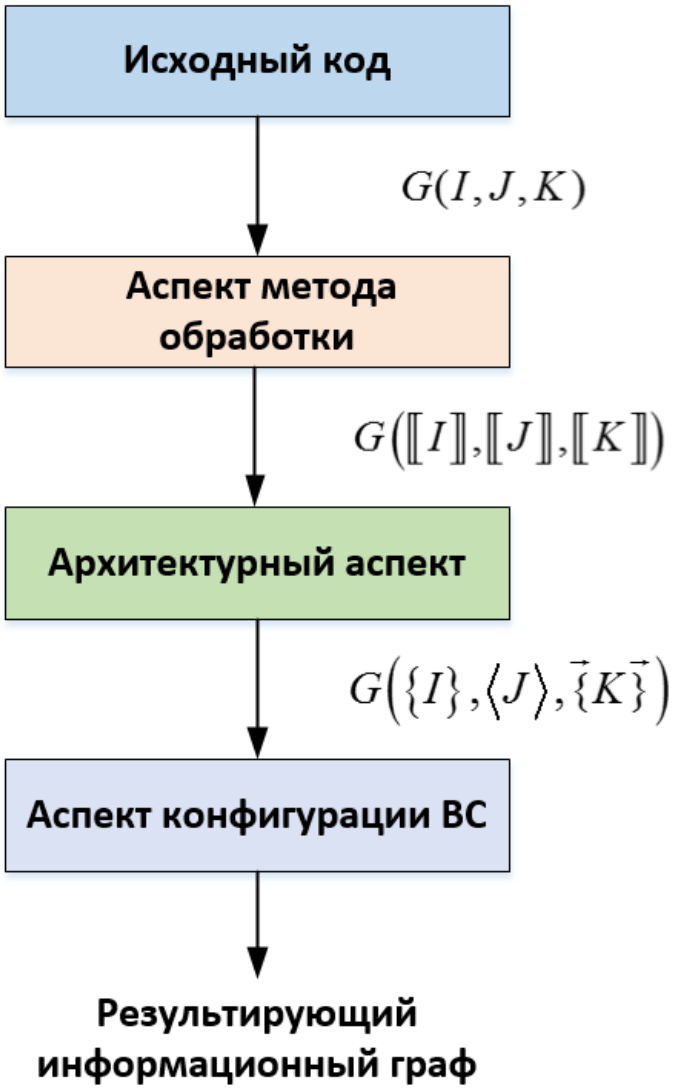
Интерфейсный раздел

```
interface::
   c: input(cmp1);
   k: output(asp2);
   d: extern;
end(interface);
```

Другие разделы

```
block1::
   a = v + t*k;
end(block1);
```

Структура аспектно-ориентированной программы на языке Set@l



- **Исходный код** описывает информационный граф *G* как отношение между совокупностями строк, столбцов и итераций обработки матрицы *I*, *J* и *K*.
- **Аспект метода обработки** задает различные методы обработки матрицы, разбивая *I*, *J* и *K* на подмножества.
- **Архитектурный аспект** определяет метод обработки в соответствии с архитектурой ВС, описывая типизацию *I*, *J* и *K* и связывая параметры их разбиений с параметрами конфигурации.
- **Аспект конфигурации ВС** конкретизирует разбиения *I*, *J* и *K*, подставляя конкретные числовые значения параметров конфигурации.

Множества в языке Set@l

Множество – основной объект языка программирования Set@l

Объявление множества:

```
set(<имя множества>);
```

```
set(P);   set(A,B,C);
```

Задание элементов множества:

- перечисление элементов:

```
A=set(1,2,4,8,16,32);
```

- задание диапазона:

```
<имя множества>=<тип>(<1-й эл.>,<2-й эл.>...<последний эл.>);
```

```
A=set(1,3...q);    A=set(1...q);
```

- реляционные выражения:

```
<имя множества>=<тип>( <переменная> | <предикат> );
```

```
A=set(p | p in N and mod(p,2)=1);
B=set(p | p in N and mod(p,2)=0);
C=set(p | p in N and p>=a and p<=b);
```

Операции над множествами и их элементами

Обращение к элементам множества:

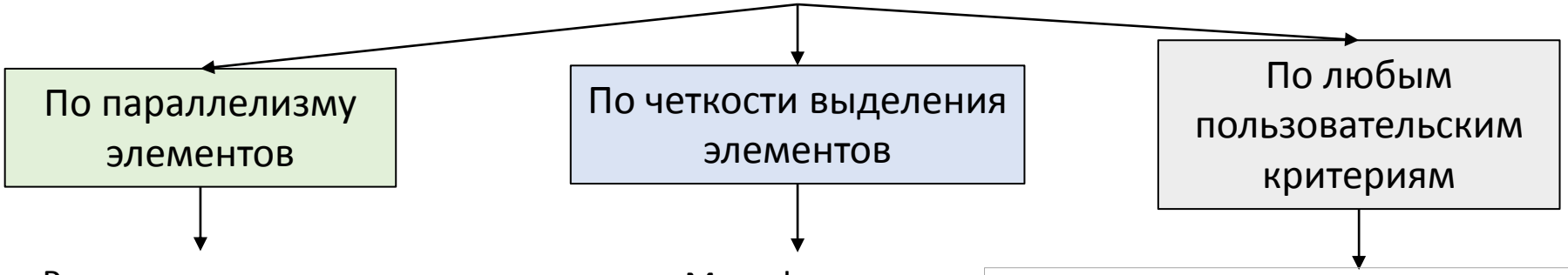
```
P=set(1...15);  P(5)=5;  P(15)=15;

Q=set(A,B);  A=set(1...5);  B=set(10...50);
Q(1,2)=2;  Q(2,4)=40;
```

Операция	Формат	Операция	Формат
Квантор существования (∃)	`exists`	Принадлежность ($a \in A$)	`a in A`
Квантор всеобщности (∀)	`forall`	Объединение (A ∪ B)	`union(A,B)`
Квантор существования и единственности (∃!)	`exists!`	Пересечение (A ∩ B)	`inter(A,B)`
Конъюнкция (A & B)	`A and B`	Разность (A \ B)	`dif(A,B)`
Дизъюнкция (A ∨ B)	`A or B`	Произведение множеств (A × B)	`prod(A,B)`
Инверсия (! A)	`not A`	Подмножество (A ⊂ B)	`sub(A,B)`

В языке Set@l аналогами циклов являются кванторы и реляционные конструкции:

```
(forall <переменная-итератор> in <множество> | <предикат>):
    <тело цикла>;
end(forall);
```

Типизация совокупностей в языке Set@l



Задание типа совокупности:

- при объявлении или заполнении совокупности элементами;

```
set(A);   B=set(1...p);
```

- с помощью конструкции `type`.

```
type(<имя совокупности>)='<тип совокупности>';
type(A)='set';
```

Любая совокупность в языке Set@l может иметь **несколько признаков**, классифицирующих ее по **разным** критериям.

```
{<тип 1>,<тип 2>...<тип n>}(<имя совокупности>);
{set,unordered}(A);
```

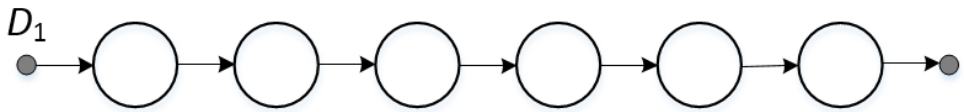
Типы совокупностей по параллелизму

Для неявного описания распараллеливания алгоритмов в языке Set@l вводится классификация совокупностей **по типу параллелизма** их элементов при обработке.

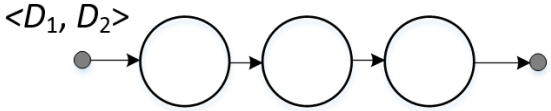
Тип совокупности	Тип обработки	Символьное обозначение	Формат описания
Множество	Параллельно-независимая	{1,2,…,p}	`par(1…p)`
Кортеж	Последовательная	[1,2,…,p]	`seq(1…p)`
Кортеж-конвейер	Конвейерная	⟨1,2,…,p⟩	`pipe(1…p)`
Множество обработки по итерациям	Параллельно-зависимая	$\vec{\{}1,2,\ldots,p\vec{\}}$	`conc(1…p)`
Неопределенный	Тип определяется в другом аспекте	[[1,2,…,p]]	`imp(1…p)`

$$G = [[G_1, G_2, G_3, G_4, G_5, G_6]];$$

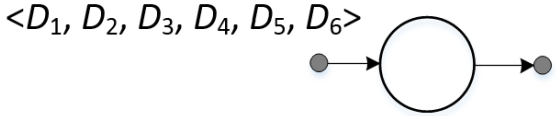
$$G = \vec{\{}G_1, G_2, G_3, G_4, G_5, G_6\vec{\}};$$



$$G = \left\langle \vec{\{}G_1, G_2, G_3\vec{\}}, \vec{\{}G_4, G_5, G_6\vec{\}} \right\rangle;$$



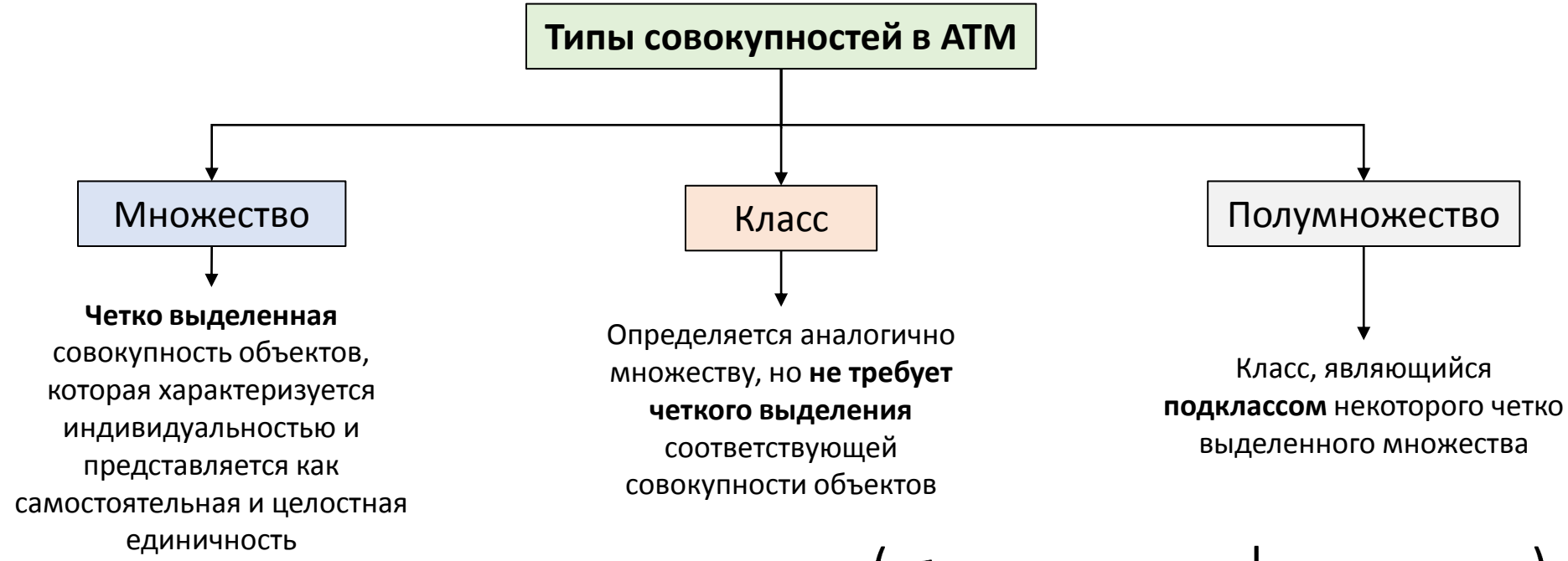
$$G = \langle G_1, G_2, G_3, G_4, G_5, G_6 \rangle;$$



Нечеткие совокупности в языке Set@l

Если аспекты **изменяют** алгоритм в процессе архитектурной адаптации, то некоторые совокупности **выделены нечетко** и не являются множествами, поэтому их невозможно описать на языке теории множеств **Кантора-Больцано**.

В языке Set@l используется **классификация** совокупностей по четкости выделения их элементов в соответствии с **альтернативной теорией множеств** П. Вопенка.



(Обезьяна Чарли, ... | ... , Ч. Дарвин)

Обезьяны Люди

Горизонт – объект, при движении в направлении которого проявляется феномен нечеткости.

Нечеткие совокупности в языке Set@l

Тип совокупности	Описание	Символьное обозначение	Ключевое слово
Множество	Четко выделенная совокупность элементов	{}	set
Полумножество	Нечетко выделенная совокупность элементов	{ ? ? } { ? ¿ }	sm
Класс	Совокупность, тип которой не может быть определен однозначно	? ¿	cls

Класс – наиболее общий и универсальный тип совокупностей в языке Set@l.

```
cls(<имя класса>);
typing(<имя класса>) : '<тип 1>' or '<тип 2>';
```

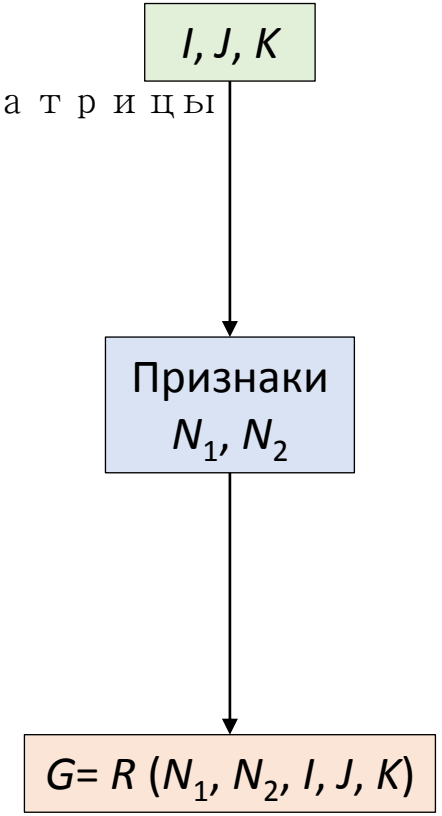
Доопределение в одном из аспектов позволяет конкретизировать **тип** и **заполнение** класса при трансляции.

Рассмотренная классификация совокупностей по четкости выделения элементов позволяет описывать разные способы реализации одного и того же алгоритма в **единой** аспектно-ориентированной программе.

Прямой ход метода Гаусса. Исходный код

```
(1)program(Gauss_forward):
(2)   interface::
(3)      G,n,m,p: output(processing,architecture);
(4)   end(interface);
(5)   data preparation::
(6)      n=<число строк в матрице>;
(7)      I=set(1...n); J=set(1...n+1); K=set(1...n);
(10)        a(1,*,*)=<ввод исходной расширенной матрицы
СЛАУ>;
(13)     end(data preparation);
(14)     graph description::
(15)        {graph,imp}(G);
(16)        attribute N1(element(k),element(i),element(j)):
(17)           a(k+1,i,j)=a(k,i,j)-a(k,i,k)/a(k,k,k)*a(k,k,j);
(18)        end(N1);
(19)        attribute N2(element(k),element(i),element(j)):
(20)           a(k+1,i,j)=a(k,i,j);
(21)        end(N2);
(22)        (forall k in K):
(23)           (forall (i,j) in prod(I,J)|i>=k+1):
(24)              G(k,i,j)=N1(k,i,j);
(25)           end(forall);
(26)           (forall (i,j) in prod(I,J)|i<k+1):
(27)              G(k,i,j)=N2(k,i,j);
(28)           end(forall);
(29)        end(forall);
(30)     end(graph description);
```

a(<итерация>,<строка>,<столбец>)



LU-разложение. Исходный код

```
(1)  int(n)=<size of matrix>;
(2)  K=set(1…n-1);                  // множество номеров итераций;
(3)  I=set(1…n);                    // множество номеров строк;
(4)  J=set(1…n);                    // множество номеров столбцов;
(5)  set(a);                        // матрица СЛАУ;
(6)  {graph,imp}(G);                // информационный граф алгоритма;
(7)  (forall i in I and j in J):
(8)    a(1,i,j)=a_init(i,j);        // загрузка исходной матрицы;
(9)  end(forall);
```

Исходные данные

```
(10) attribute LU_iter(element(k),set(s1),set(s2)):
(11)    (forall i in s1 and j in s2 | i<=k or (i>k and j<k)):
(12)      a(k+1,i,j)=a(k,i,j);
(13)     end(forall);
(14)    (forall i in s1 | i>k):
(15)      a(k+1,i,k)=a(k,i,k)/a(k,k,k);
(16)      (forall j in s2 | j>k):
(17)        a(k+1,i,j)=a(k,i,j)-a(k+1,i,k)*a(k,k,j);
(18)      end(forall);
(19)    end(forall);
(20) end(LU_iter);
```

Описание операционных вершин графа

```
(21) (forall k in K):
(22)    G(k)=LU_iter(k,I,J);
(23) end(forall);
```

Формирование полного информационного графа

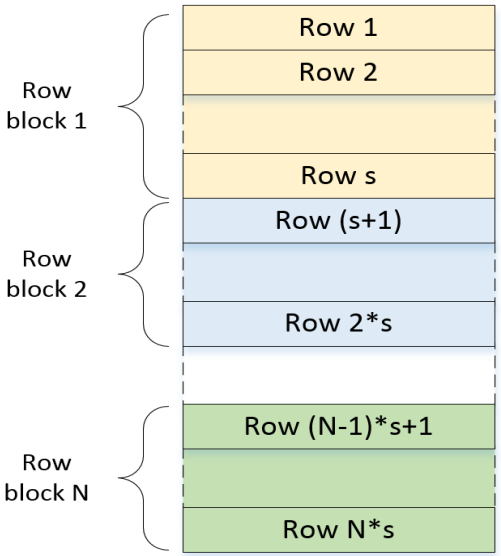
Прямой ход метода Гаусса, LU-разложение. Аспект метода обработки

```
(1) aspect(processing):
(2)     interface::
(3)         s,N,c,M,ni,T: input(architecture);
(4)         G,n,m,p: input(Gauss_forward);
(5)     end(interface);
(6)    set partition::
```

Разбиение G

1. Обработка по строкам

```
(7)       (forall k in K):
(8)          R(k,i)=(G(k,i,j)|j in J);                    // i-я строка;
(9)          RB(k,p)=imp(R(k,i)|i in ((p-1)*s+1...p*s));  // p-й блок строк;
(10)         GR(k)=imp(RB(k,p)|p in (1...N));             // подграф из строк;
(11)      end(forall);
```



$$GR_k = \left[\underbrace{\left[\left[R_{k,1} \dots R_{k,s}\right]\right]}_{RB(k,1)}, \underbrace{\left[\left[R_{k,s+1} \dots R_{k,2*s}\right]\right]}_{RB(k,2)}, \dots, \underbrace{\left[\left[R_{k,(N-1)*s+1} \dots R_{k,N*s}\right]\right]}_{RB(k,N)}\right];$$

```
// s – число строк в блоке;
// n – число строк в обрабатываемой матрице;
// N=n/s – число блоков строк в матрице;
```

$$s=1:\ GR_k = \left[\left[\left[R_{k,1}\right]\right], \left[\left[R_{k,2}\right]\right], \dots, \left[\left[R_{k,n}\right]\right]\right];$$

$$s=2:\ GR_k = \left[\left[\left[R_{k,1}, R_{k,2}\right]\right], \left[\left[R_{k,3}, R_{k,4}\right]\right], \dots, \left[\left[R_{k,n-1}, R_{k,n}\right]\right]\right];$$

$$s=n:\ GR_k = \left[\left[\left[R_{k,1}, R_{k,2}, \dots, R_{k,n-1}, R_{k,n}\right]\right]\right];$$

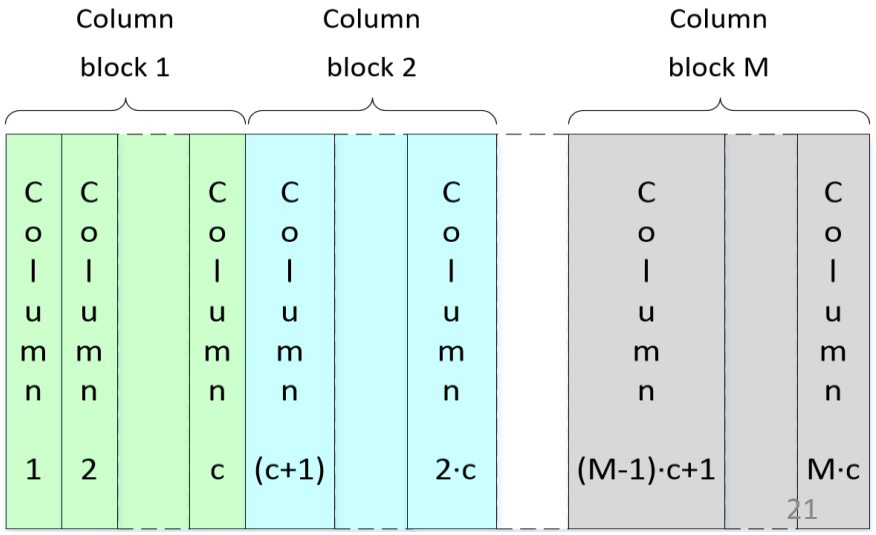
Прямой ход метода Гаусса, LU-разложение. Аспект метода обработки

2. Обработка по столбцам

```
(12)    (forall k in K):
(13)      C(k,j)=(G(k,i,j)|i in I);                    // j-й столбец;
(14)      CB(k,q)=imp(C(k,j)|j in ((q-1)*c+1...q*c));  // q-й блок столбцов;
(15)      GC(k)=imp(CB(k,q)|q in (1...M));             // подграф из столбцов;
(16)    end(forall);
```

$$GC_k = \left[\underbrace{[C_{k,1} \dots C_{k,c}]}_{CB(k,1)}, \underbrace{[C_{k,c+1} \dots C_{k,2*c}]}_{CB(k,2)}, \dots, \underbrace{[C_{k,(M-1)*c+1} \dots C_{k,M*c}]}_{CB(k,M)}\right];$$

```
// c – число столбцов в блоке;
// m – число столбцов в матрице;
// M=m/c – число блоков столбцов в матрице;
```



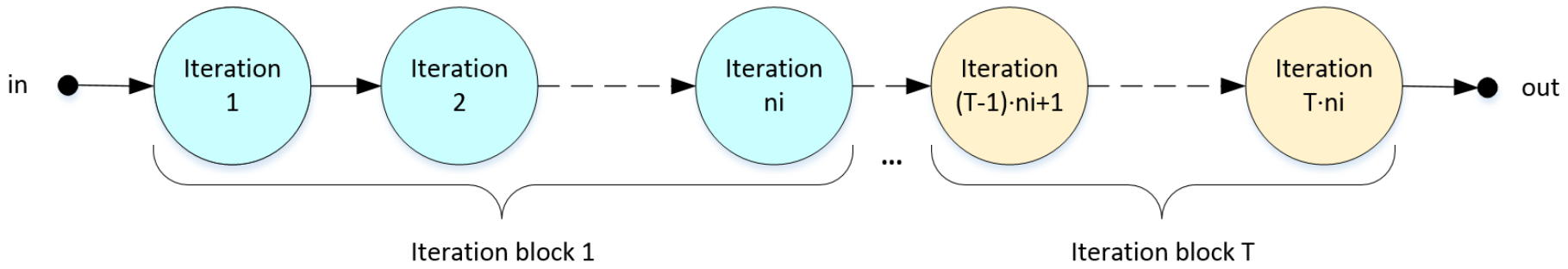
Прямой ход метода Гаусса, LU-разложение. Аспект метода обработки

3. Обработка по итерациям

```
(17)    I(k)=(G(k,i,j)|i in I and j in J);          // k-я итерация;
(18)    IB(l)=imp(I(k)|k in ((l-1)*ni+1...l*ni));   // l-й блок итераций;
(19)    GI=imp(IB(l)|l in (1...T));                 // граф из итераций;
```

$$GI = \left[\underbrace{[\![I_1 \dots I_{ni}]\!]}_{IB(1)}, \underbrace{[\![I_{ni+1} \dots I_{2*ni}]\!]}_{IB(2)}, \dots, \underbrace{[\![I_{(T-1)*ni+1} \dots I_{T*ni}]\!]}_{IB(M)}\right];$$

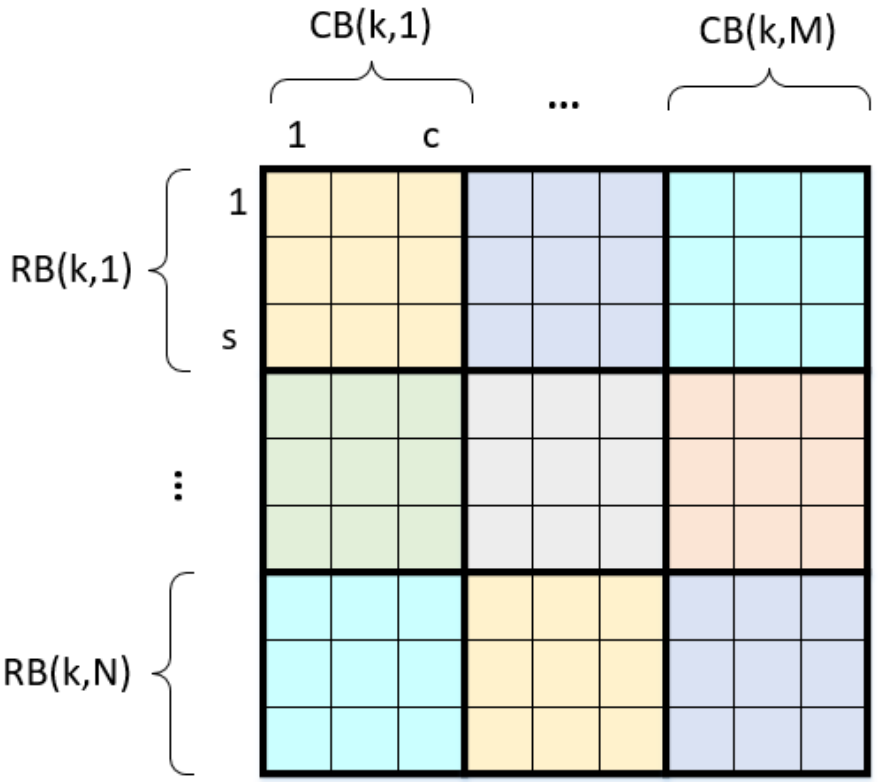
```
// ni – число итераций в блоке;
// p – общее число итераций, необходимых для обработки матрицы;
// T=p/ni – число блоков итераций;
```



Прямой ход метода Гаусса, LU-разложение. Аспект метода обработки

4. Обработка по клеткам

Разбиение на **строки** + разбиение на **столбцы**




```
// RB(k,i) – i-й блок строк;
// CB(k,j) – j-й блок столбцов;
// s – число строк в клетке;
// c – число столбцов в клетке;
```

```
// n – число строк в матрице;
// m – число столбцов в матрице;
// N=n/s – число клеток по строкам;
// M=m/c – число клеток по столбцам;
```

Прямой ход метода Гаусса, LU-разложение

Архитектурный аспект

Архитектурный аспект задает **типизацию** совокупностей и связывает параметры их **разбиений** с соответствующими архитектуре параметрами **конфигурации** ВС

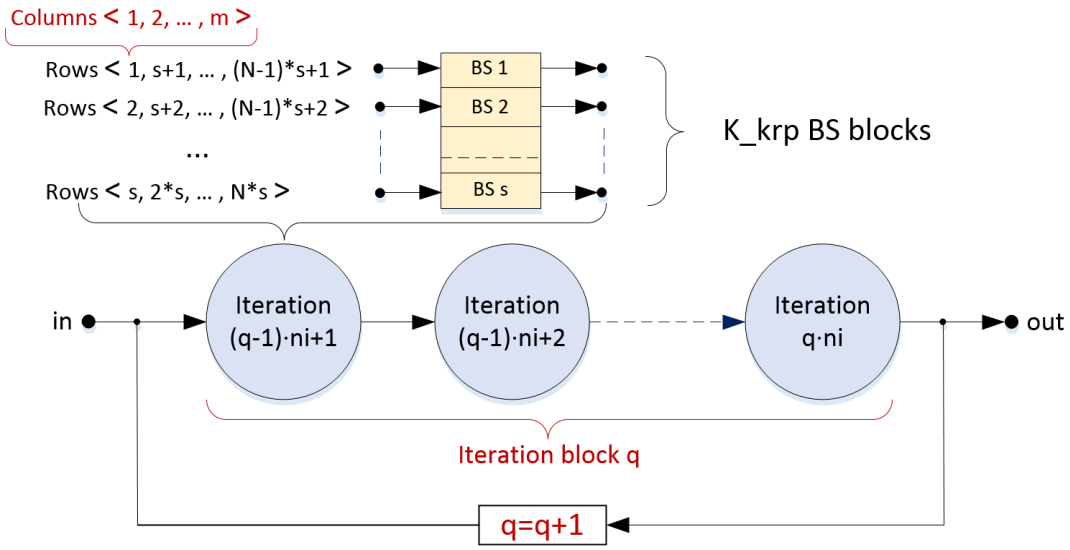
```
(1)  aspect(architecture):
(2)     interface:
(3)          GR,GC,GI: input(processing);
(4)          R,R0,K_krp,q1,q2,architecture_type: input(configuration);
(5)          G:n,m,p: input(Gauss_forward);
(6)          G: output(Gauss_forward);
(7)    end(interface);
```

РВС → Распараллеливание по строкам/итерациям в зависимости от числа КРП и доступного вычислительного ресурса

Мультипроцессорная ВС → Распараллеливание по клеткам в зависимости от числа процессоров

Прямой ход метода Гаусса, LU-разложение. Архитектурный аспект

1. ВС с реконфигурируемой архитектурой




```
// Строки:
(8)   s=K_krp;
(9)   N=n/s;
(10)  type(RB(k,p))='par';
(11)  type(GR(k))='pipe';

// Столбцы:
(12)  c=m;
(13)  M=1;
(14)  type(CB(k,q))='pipe';
(15)  type(GC(k))='pipe';

// Итерации:
(16) ni=min( p, floor(R/s/R0) );
(17) T=p/ni;
(18) type(IB(1))='conc';
(19) type(GI)='pipe';
```

$$GR_k = \langle \{R_{k,1} \dots R_{k,s}\}, \dots, \{R_{k,(N\text{-}1)*s+1} \dots R_{k,N*s}\} \rangle;$$

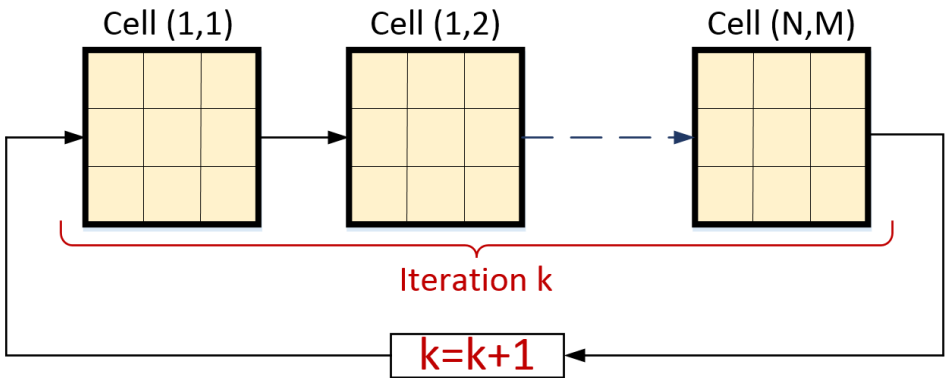
$$GC_k = \langle C_1, C_2, \dots, C_m \rangle;$$

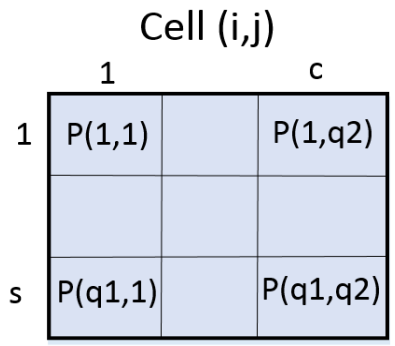
$$GI = \langle \{\vec{I}_1 \dots I_{ni}\}, \{\vec{I}_{ni+1} \dots I_{2*ni}\}, \dots, \{\vec{I}_{(T\text{-}1)*ni+1} \dots I_{T*ni}\} \rangle;$$

- Распараллеливание **по строкам** в зависимости от числа **каналов КРП** (метод Гаусса), **конвейерная** обработка **строк** (LU-разложение)
- **Столбцы** обрабатываются **конвейерно**
- Распараллеливание **по итерациям** определяется доступным **вычислительным ресурсом**

Прямой ход метода Гаусса, LU-разложение. Архитектурный аспект

2. Мультипроцессорная ВС






```
// Строки:
(8)   s=q1;
(9)   N=n/s;
(10)  type(RB(k,p))='par';
(11)  type(GR(k))='seq';

// Столбцы:
(12)  c=q2;
(13)  M=m/c;
(14)  type(CB(k,q))=' par' ;
(15) type(GC(k))=' seq' ;

// И т е р а ц и и:
(16) ni=p;
(17) T=1;
(18) type(IB(l))=' seq' ;
(19) type(GI)=' seq' ;
```

$$GR_k = \left[\left\{ R_{k,1} \dots R_{k,s} \right\}, \dots, \left\{ R_{k,(N\text{-}1)*s+1} \dots R_{k,N*s} \right\} \right];$$

$$GC_k = \left[\left\{ C_{k,1} \dots C_{k,c} \right\}, \dots, \left\{ C_{k,(M\text{-}1)*c+1} \dots C_{k,M*c} \right\} \right];$$

$$GI = \left[I_1, I_2, \dots, I_p \right];$$

- Распараллеливание **по клеткам** в зависимости от **числа процессоров** в МП ВС
- Итерации реализуются **последовательно**

Прямой ход метода Гаусса, LU-разложение. Аспект конфигурации ВС

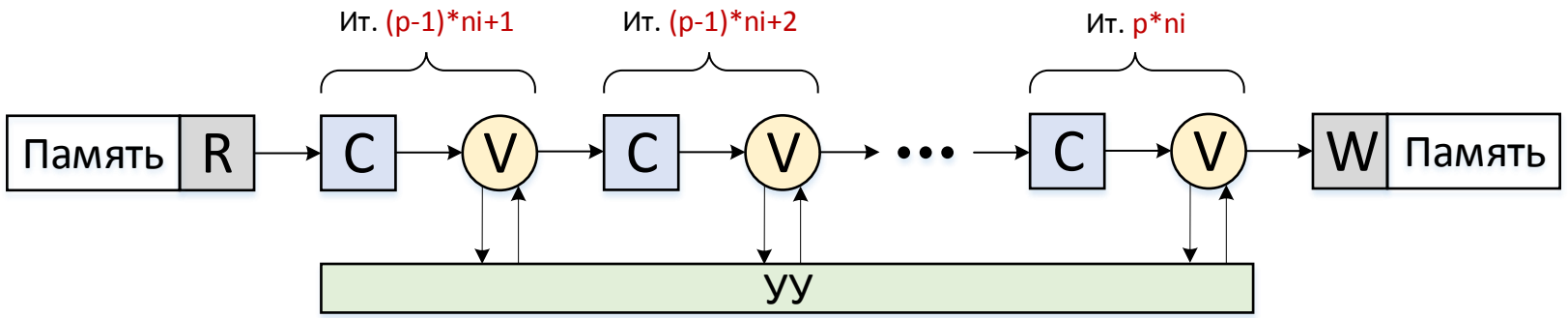
Данный аспект определяет числовые значения **параметров конфигурации** ВС.

```
(1)  aspect(configuration):

(2)     interface:
(3)          R,R0,K_krp,q1,q2,architecture_type: output(architecture);
(4)     end(interface);

(5)     architecture_type=<тип архитектуры ВС>;

(6)     case(architecture_type=’RCS’):
(7)          R=<доступный вычислительный ресурс>;
(8)          R0=<выч. ресурс, необходимый для
                реализации минимального
(9)                базового подграфа>;
(9)          K_krp=<количество КРП в РВС>;
(10)    end(case);

(11)    case(architecture_type=’MP’):
(12)         q1=<размерность матрицы процессоров по
строкам>;
(13)         q2=<размерность матрицы процессоров по
столбцам>;
(14)    end(case);
```

Метод Якоби. Способы реализации

Способы реализации алгоритма решения СЛАУ методом Якоби на РВС:

1. Проверка условия завершения вычислений на каждой итерации



C и V – блоки **пересчета** и **проверки** условия завершения вычислений $err \le \delta$.

K – совокупность **номеров итераций** алгоритма.

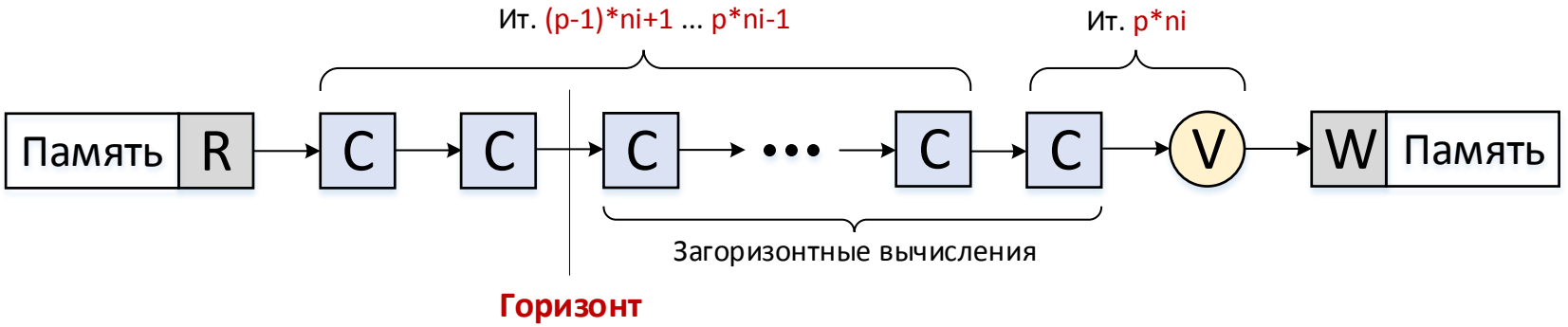
K – **множество**, так как известны его первый элемент, принцип заполнения и номер последней итерации I_m, который неизвестен заранее, но четко определяется условием завершения вычислений.

$$K = \mathbf{set}\left(k \mid k \in \mathbb{N} \ \& \ \left(k = 1 \ \mathbf{or} \ err(k-1) > \delta\right)\right);$$

$$K = \left\langle \vec{\{1 \ldots ni\}}, \vec{\{ni+1 \ldots 2 \cdot ni\}}, \ldots, \vec{\{(T-1) \cdot ni + 1 \ldots I_m\}} \right\rangle;$$

Метод Якоби. Способы реализации

2. Проверка условия завершения вычислений после нескольких итераций пересчета



$$K^* = \left\langle \vec{\{1 \ \ldots \ ni\}}, \vec{\{ni+1 \ \ldots \ 2 \cdot ni\}}, \ \ldots \ , \vec{\{(T-1) \cdot ni + 1 \ \ldots \ T \cdot ni\}} \right\rangle;$$

Множества *K* и *K** описывают один и тот же математический объект только в том случае, когда условие выполняется на итерации с номером *T·ni*.

$$K^* = K \cup K_{\text{заг}}$$

С точки зрения АТМ совокупность *K* является **полумножеством** – классом, связанным с *K** отношением включения.

$$K^*_{sub}(k) = \mathbf{set}\left(\ k_1 \ldots k_2 \ \mid \ k_1 = (k-1) \cdot ni + 1 \ \ \& \ \ k_2 = k \cdot ni \ \right);$$

$$K^* = \mathbf{set}\left(\ K^*_{sub}(k) \ \mid \ k \in \mathbb{N} \ \ \& \ \ \left(k = 1 \ \ \mathbf{or} \ \ err\left((k-1) \cdot ni \right) > \delta \right) \ \right);$$

$$\mathbf{sm}(K) \subseteq \mathbf{set}(K^*);$$

Заключение

Использование языка Set@l открывает **принципиально новые возможности** для оперативного портирования сложных программных комплексов между ВС с различными архитектурами, в том числе **гибридными** и **реконфигурируемыми**.

Программирование на языке Set@l:

- **Исходный код** описывает информационный граф решаемой задачи в архитектурно-независимой форме в виде совокупностей, их признаков и отношений между ними.
- Подключая те или иные **аспекты**, возможно описать любые преобразования исходного кода и задать множество вариантов реализации алгоритма на разных архитектурах и конфигурациях ВС.
- Аспекты архитектуры и разбиения множеств могут быть **обобщены** для широкого класса прикладных задач.
- Для некоторых задач возможно описать переход между **разными алгоритмами** их решения в одной аспектно-ориентированной программе.

БЛАГОДАРЮ ЗА ВНИМАНИЕ!

Если остались вопросы:
dordopulo@superevm.ru