НИЦ СЭ и НК

АСПЕКТНО-ОРИЕНТИРОВАННЫЙ ПОДХОД К АРХИТЕКТУРНО-НЕЗАВИСИМОМУ ПРОГРАММИРОВАНИЮ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ – язык SET@L

Проблемы программирования параллельных ВС

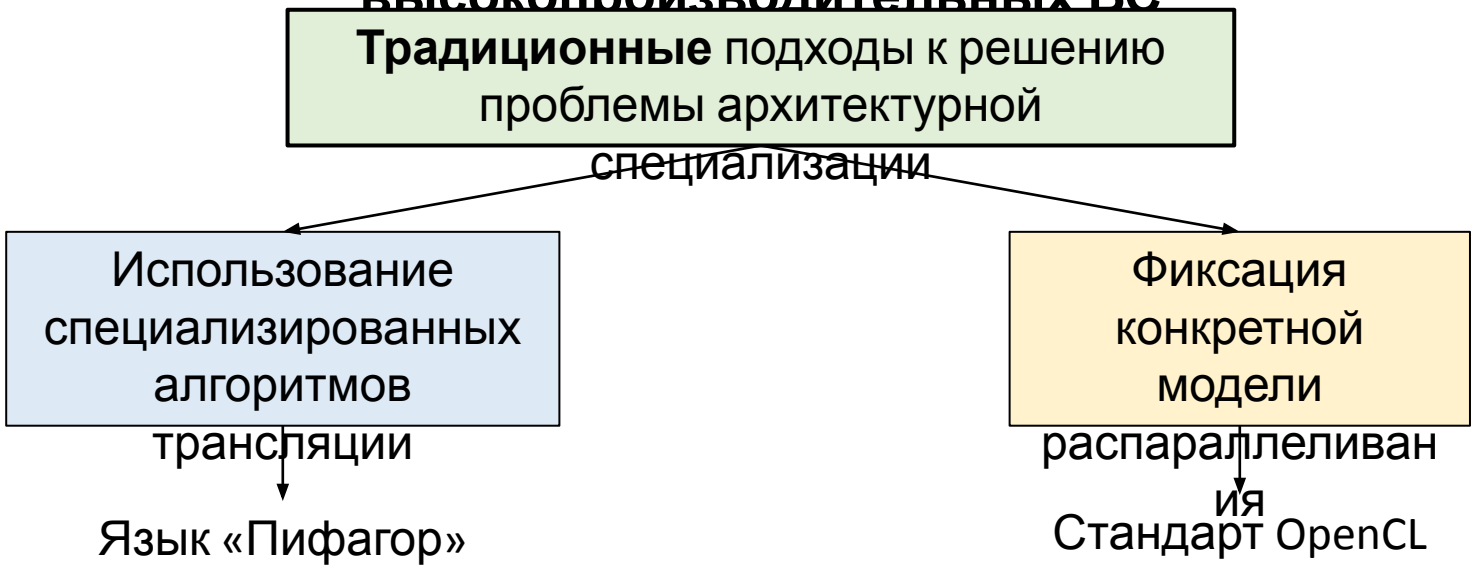
Объединение **процессоров** с вычислительными устройствами **других типов** – одно из перспективных архитектурных решений, позволяющих повысить производительность современных вычислительных систем (ВС).

Программное обеспечение для **гибридных ВС** должно сочетать описания параллельных, конвейерных и процедурных фрагментов вычислений, что приводит к **усложнению** его разработки и **повышению затрат** временных и материальных ресурсов.

В настоящее время в области программирования высокопроизводительных ВС **не существует** эффективных методов и средств описания **архитектурно-независимых** программ.

В традиционных языках программирования **математическая сущность** решения прикладной задачи и его **декомпозиция** описываются **неделимыми** фрагментами кода. Поэтому изменение каких-либо особенностей распараллеливания, связанное с реализацией алгоритма на ВС с другой архитектурой, фактически требует разработки **новой программы**.

Современные средства программирования высокопроизводительных ВС



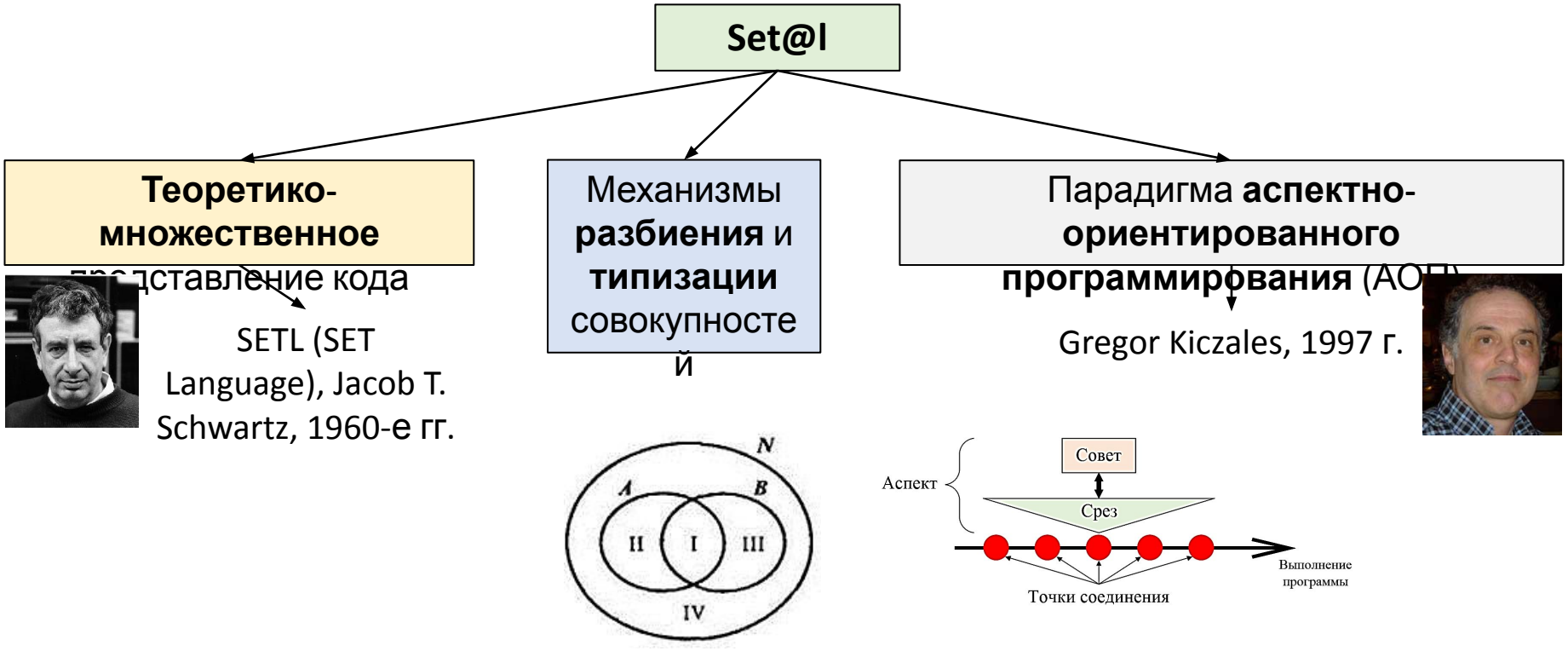
Большинство проблем, связанных с программированием реконфигурируемых ВС на основе программируемых логических интегральных схем (ПЛИС), решено в высокоуровневом языке программирования **COLAMO**.

В COLAMO распараллеливание описывается в **неявной форме** путем объявления **типов доступа к массивам** 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**.

Язык архитектурно-независимого программирования ВС Set@l

Дальнейшим развитием идей COLAMO является язык **архитектурно-независимого** программирования ВС **Set@l**.



Set@l позволяет свести портацию ПО к **формированию** соответствующих **аспектов** архитектуры и конфигурации ВС, в которых описываются **разбиение** и **типизация** ключевых совокупностей алгоритма. При этом исходный код, определяющий математическую сущность решения прикладной задачи, остается **неизменным**.

Аспектно-ориентированный подход к программированию ВС

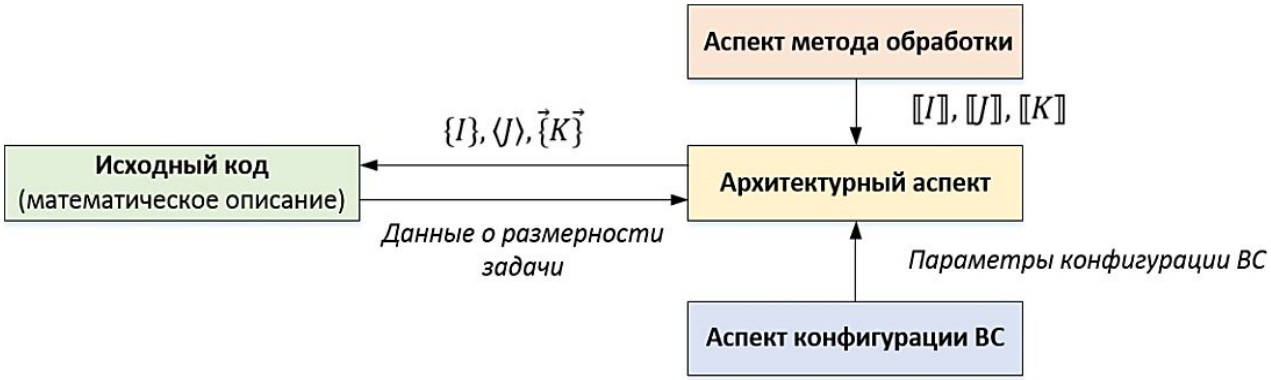
В существующих языках программирования ВС **архитектурная специализация ПО** обусловлена тем, что **алгоритм** решения прикладной задачи и конкретный **вариант** его **распараллеливания** описываются **неделимым кодом** (что приводит к **спутыванию** в терминах АОП) и **рассредоточены** по тексту программы (что приводит к **разбрасыванию** в терминах АОП).

Фрагменты программы, задающие распараллеливание и другие особенности реализации, выделяются в отдельные модули – **аспекты**, что позволяет описывать алгоритм решения прикладной задачи **независимо** от архитектурных особенностей вычислительной системы, на которой будет осуществляться ее решение.

В таком случае исходный код программы является **архитектурно-независимым**, а код аспектов определяет **декомпозицию** прикладной задачи для реализации ее решения на ВС с определенными архитектурой и конфигурацией.

Анализируя созданную пользователем разметку, **транслятор-препроцессор**

Принципы программирования на языке Set@l



- **Исходный код** описывает решение прикладной задачи в виде информационного графа на языке теории множеств и реляционного исчисления, формируя универсальное архитектурно-независимое представление алгоритма.
- **Аспекты**, направленные на адаптацию алгоритма к архитектуре и конфигурации конкретной ВС, разделяют исходные множества вершин обработки и данных итераций на особые совокупности, для каждой из которых задаются тип обработки и разбиение.
- **Аспект метода обработки** определяет логику различных вариантов разбиения матрицы (например, по строкам, столбцам, клеткам или итерациям) путем задания обобщенных совокупностей неопределенного типа и размерности.
- **Архитектурный аспект**, используя данные о размерности задачи из исходного кода и конкретные числовые параметры конфигурации ВС из **аспекта конфигурации**, преобразует или задает тип совокупностей в соответствии с особенностями архитектуры ВС для эффективной параллельно-конвейерной обработки.

Классификация совокупностей по определенности, используемая в языке программирования Set@l

По умолчанию все совокупности в языке Set@l считаются **четко выделенными множествами**, однако существуют некоторые задачи, решение которых не может быть описано в архитектурно-независимой форме без привлечения совокупностей других типов.

В языке Set@l используется типизация совокупностей, предложенная П. Вопенка в рамках **альтернативной теории множеств**.

Примером задачи, в которой используются объекты альтернативной теории множеств, является решение СЛАУ итерационным методом Якоби, в рамках которого совокупность итераций определяется как **класс** и может представлять собой **полумножество** или **множество** в зависимости от выбранного варианта реализации алгоритма.

Тип совокупности	Описание	Символьное обозначение	Ключевое слово
Множество	Четко выделенная совокупность элементов		**set**
Полумножество	Частично определенная совокупность элементов		**sm**
Класс	Полностью неопределенная совокупность элементов,		**cls**

Типы совокупностей по параллелизму

В отличие от других языков программирования на основе теории множеств (например, языка SETL), в Set@l вводится **классификация совокупностей по различным критериям**.

С точки зрения архитектурной независимости, основополагающим критерием типизации множеств является **характер параллелизма их элементов при обработке**.

Конструкция **type**:

type(<*имя совокупности*>**)=**'<*тип параллелизма*>';

Тип совокупности	Тип обработки	Символьное обозначение	Формат описания
Множество	Параллельно-независимая		**set(**1…p**)**
Кортеж	Последовательная		**seq(**1…p**)**
Кортеж-конвейер	Конвейерная		**pipe(**1…p**)**
Множество обработки по итерациям	Параллельно-зависимая		**conc(**1…p**)**
Неопределенный	Тип определяется в другом аспекте		**imp(**1…p**)**

Прямой ход метода Гаусса. Исходный код

program (*Gauss_forward*):

interface :: *// интерфейсный раздел;*

I, J, K **: input** (*architecture*); *// входные параметры – строки, столбцы, итерации;*

n, m, p : **output** (*architecture*); *// выходные параметры – размерность задачи (арх. аспект);*

end (interface);

data preparation :: *// раздел подготовки исходных данных;*

n=<*размерность матрицы по строкам*>;

m=n+1; *// размерность матрицы по столбцам;*

p=n; *// количество итераций обработки матрицы в прямом ходе;*

a(1,*,*)=<*ввод элементов исходной матрицы*>;

end (data preparation);

computing :: *// вычислительный раздел;*

(**forall** k **in** K) :

(**forall seq**(i,j) **in prod(**I,J**)** | i>=(k+1)):

a(k+1,i,j)=a(k,i,j)-a(k,k,j)*a(k,i,k)/a(k,k,k); *// пересчет;*

end (forall);

(**forall seq**(i,j) **in prod(**I,J**)** | i<(k+1)):

a(k+1,i,j)=a(k,i,j); *// сохранение;*

end (forall);

end (forall);

end (computing);

Прямой ход метода Гаусса

Аспект метода обработки

aspect (processing):

interface :: // *интерфейсный раздел;*

s, N, c, M, ni, T : **input** (*architecture*); // *входные параметры;*

I, J, K : **output** (*architecture*); // *выходные параметры;*

end (interface);

1. *Метод обработки по строкам*

I=**imp**(BL(i) | BL(i)=**imp**((i-1)*s+1 … i*s) **and** i **in** **seq**(1…N));

// BL(i) – *набор номеров строк, попадающих в* i-*й блок;*

// s – *число строк в блоке;*

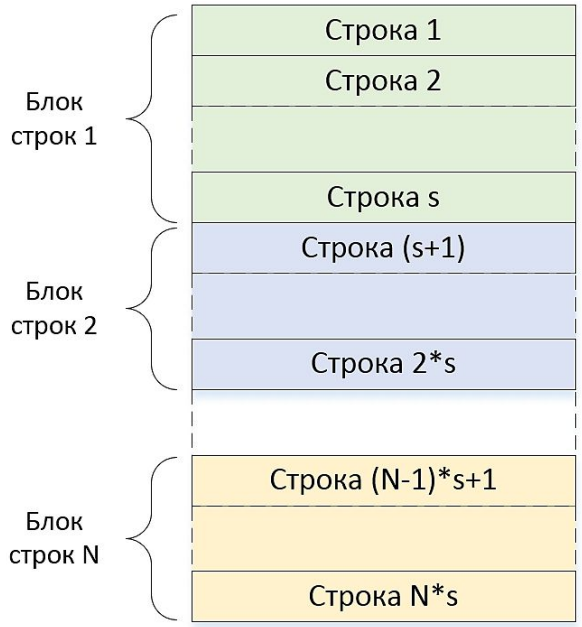
// n – *число строк в обрабатываемой матрице;*

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

// *Индексы строк:*

// I=⟦⟦1…s⟧, ⟦s+1 … 2*s⟧, … ,⟦(N–1)*s+1 … N*s⟧⟧;

// BL_1 BL_2 BL_N



Прямой ход метода Гаусса

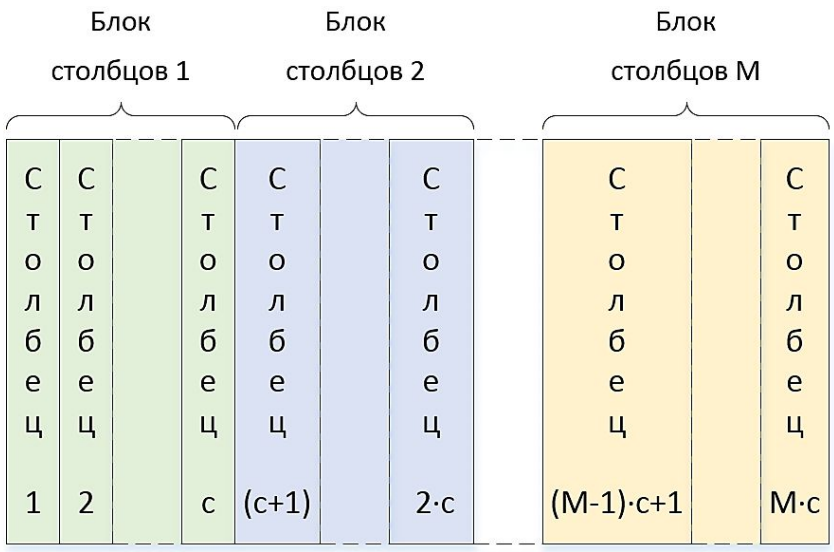
Аспект метода обработки

2. *Метод обработки по столбцам*

J=**imp**(BC(j) | BC(j)=**imp**((j-1)*c+1 … j*c) **and** j **in seq**(1…M));

// BC(j) – *совокупность номеров столбцов, попадающих в j-й блок;*
// c – *число столбцов в блоке;*
// m – *число столбцов в матрице;*
// M=m/c – *число блоков столбцов в обрабатываемой матрице;*

// *Индексы столбцов:*
// J=⟦⟦1…c⟧, ⟦c+1 … 2∗c⟧, … , ⟦(M–1)∗c+1 … M∗c⟧⟧;
// BC_1 BC_2 BC_M



Прямой ход метода Гаусса

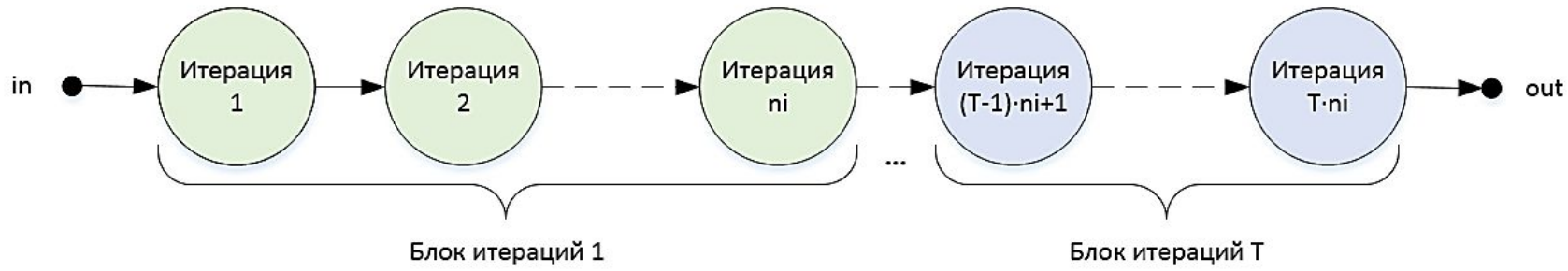
Аспект метода обработки

3. *Метод обработки по итерациям*

K=imp(BI(k) | BI(k)=conc((k-1)*ni+1 … k*ni) and k in seq(1…T));

// BI(k) – *совокупность номеров итераций в* k-*м блоке;*
// ni – *число итераций в блоке;*
// p – *общее число итераций, необходимое для обработки матрицы;*
// T=p/ni – *число блоков итераций;*

// *Индексы итераций:*
// $K=[\![\ \vec{\{}1 \ldots ni\vec{\}}, \vec{\{}ni+1 \ \ldots \ 2*ni\vec{\}}, \ldots, \vec{\{}(T-1)*ni+1 \ \ldots \ T*ni\vec{\}}\ ]\!]$;
// $\quad\quad BI^{(1)} \quad\quad BI^{(2)} \quad\quad\quad\quad BI^{(T)}$



Прямой ход метода Гаусса

Аспект метода обработки

4. *Метод обработки по клеткам*

I=imp(BL(i) | BL(i)=imp((i-1)*s+1 … i*s) and i in seq(1…N));

J=imp(BC(j) | BC(j)=imp((j-1)*c+1 … j*c) and j in seq(1…M));

// BL(i) – i-*й блок строк;*

// BC(j) – j-*й блок столбцов;*

// s – *число строк в клетке;*

// c – *число столбцов в клетке;*

// n – *число строк в обрабатываемой матрице;*

// m – *число столбцов в обрабатываемой матрице;*

// N=n/s – *число клеток по строкам;*

// M=m/c – *число клеток по столбцам;*

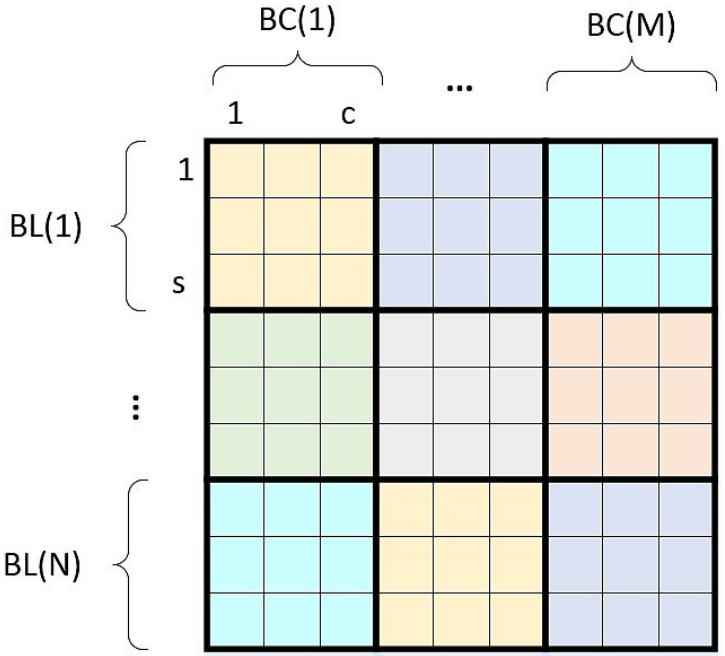
// *Индексы строк и столбцов:*

// I=⟦⟦1…s⟧, ⟦s+1 … 2∗s⟧, … ,⟦(N–1)∗s+1 … N∗s⟧⟧;

// BL_1 BL_2 BL_N

// J=⟦⟦1…c⟧, ⟦c+1 … 2∗c⟧, … , ⟦(M–1)∗c+1 … M∗c⟧⟧;

// BC_1 BC_2 BC_M



Прямой ход метода Гаусса

Архитектурный аспект

aspect (architecture):

interface :: *// интерфейс архитектурного аспекта;*

I, J, K : **input** (*processing*); *// входные параметры;*

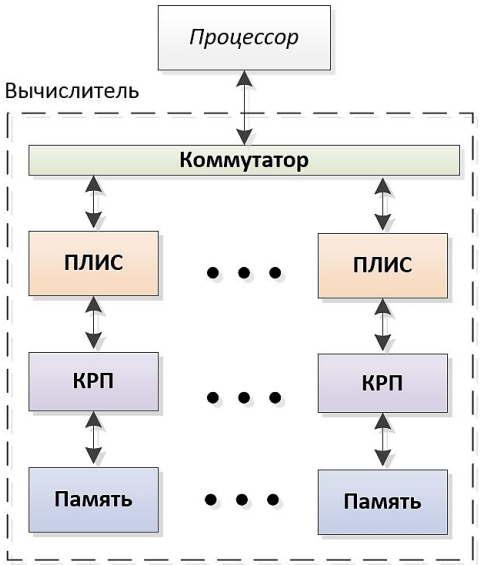
R, R0, K_krp, q1, q2, architecture_type : **input** (*configuration*); *// входные параметры;*

n, m, p : **input** (*Gauss_forward*); *// входные параметры;*

I, J, K: **output** (*Gauss_forward*); *// выходные параметры;*

Прямой ход метода Гаусса. Архитектурный аспект

1. *Архитектура с независимым полем ПЛИС* (РВС НИЦ СЭ и НК)



// Строки (I):

s=K_krp; *// число параллельно обрабатываемых строк //ограничено кол-вом КРП;*

N=n/s; *// число блоков строк в обрабатываемой матрице; //n – число строк в матр.;*

type(I)=’pipe’; *// блоки строк обрабатываются конвейерно;*

type(BL(i)**)=’par’**; *// строки в блоке обрабатываются параллельно;*

// Столбцы (J):

J=**pipe**(1…m); *// столбцы обрабатываются конвейерно;*

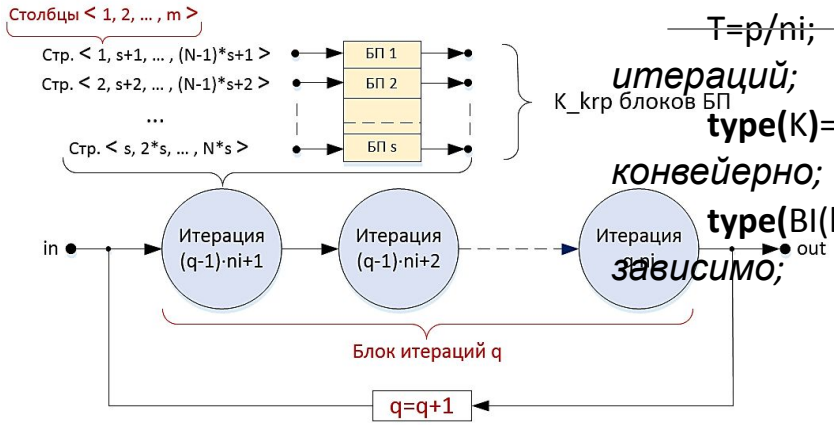
// Итерации (K);

ni=**min(** p, **floor**(R/s/R0) **)**; *// число итераций в блоке; R – доступный // выч. ресурс; R0 – ресурс, необходимый для реализации // минимального базового подграфа;*

T=p/ni; *// количество блоков итераций; p – число итераций;*

type(K**)=’pipe’**; *// блоки итераций обрабатываются конвейерно;*

type(BI(k)**)=‘conc’**; *// итерации в кадре реализуются параллельно-зависимо;*



Получаемое разбиение:

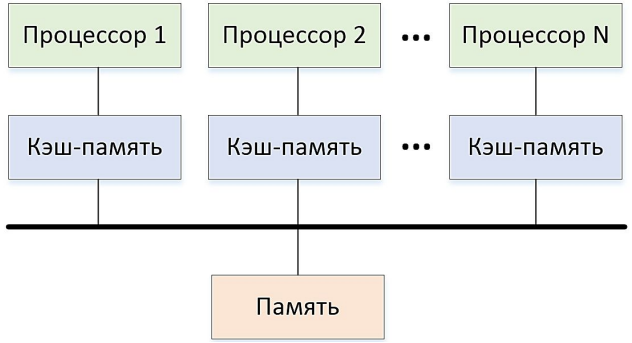
I=⟨ {1…s}, {s+1 … 2*s}, … , {(N−1)*s+1 … N*s} ⟩;

J=⟨ 1…m ⟩;

K=⟨ {1…ni}, {ni+1 … 2*ni}, … , {(T−1)*ni+1 … T*ni} ⟩;

Прямой ход метода Гаусса. Архитектурный аспект

2. Мультипроцессорная архитектура



// Строки (I):

s=q1; *// число строк в клетке;* q1 *– размерность матрицы проц. по строкам;*

N=n/s; *// число клеток по строкам;* n *– количество строк;*

type(BL(i)**)=’par’**; *// строки в клетке обрабатываются параллельно;*

type(I**)=’seq’**; *// клетки обрабатываются последовательно;*

// Столбцы (J):

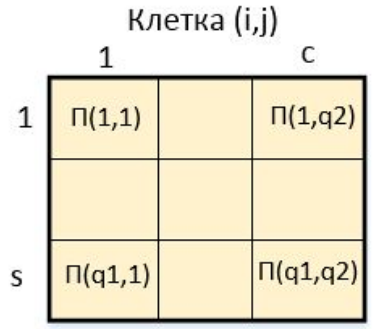
c=q2; *// число строк в клетке;* q2 *– размерность матрицы процессоров*

// по столбцам;

M=m/c; *// число клеток по строкам;* m *– количество столбцов;*

type(BC(j)**)=’par’**; *// столбцы в клетке обрабатываются параллельно;*

type(J**)=’seq’**; *// клетки обрабатываются последовательно;*



// Итерации (K):

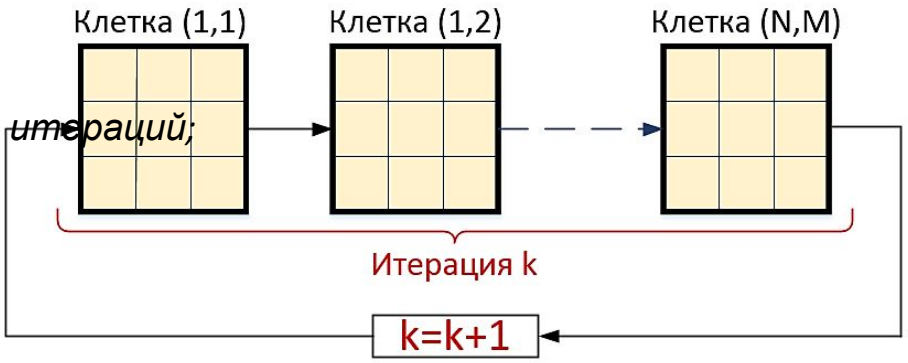
K=**seq(**1…p**)**; *// кортеж итераций; p – количество итераций;*

Получаемое разбиение:

I=[{1…s}, {s+1 … 2*s}, … , {(N–1)*s+1 … N*s}];

J=[{1…c}, {c+1 … 2*c}, … , {(M–1)*c+1 … M*c}];

K=[1…p];



Прямой ход метода Гаусса

Аспект конфигурации ВС

aspect (configuration):

---

interface :: // *интерфейс аспекта конфигурации ВС;*

R, R0, K_krp, q1, q2, architecture_type : **output** (*architecture*);

end (interface);

---

architecture_type=<*ввод типа архитектуры*>;

---

case (architecture_type='**RCS**') **:** // *архитектура с независимым полем ПЛИС (РВС НИЦ СЭ и НК);*

R=<*доступный вычислительный ресурс*>;

R0=<*вычислительный ресурс, необходимый для реализации мин. базового подграфа*>;

K_krp=<*количество каналов КРП в ВС*>;

---

end (case);

case (architecture_type='**MP**') **:** // *мультипроцессорная архитектура;*

q1=<*размерность матрицы процессоров по строкам*>;

q2=<*размерность матрицы процессоров по столбцам*>;

---

end (*multiprocessor*);

Заключение

В отличие от других средств программирования высокопроизводительных ВС, в языке Set@l решаемая вычислительная задача представляется не в виде **жестко определенных** с точки зрения параллелизма **массивов данных** и **команд**, а в виде **множеств**, их **признаков** и **отображений**.

Задавая различные варианты **разбиения** множеств и **классифицируя** их по различным признакам, можно описать алгоритм решения задачи в **архитектурно-независимой форме** и адаптировать его к любой архитектуре и конфигурации ВС с использованием **системы аспектов**.

С целью снятия архитектурных ограничений, характерных для традиционных языков программирования ВС, в языке Set@l введен механизм **типизации множеств**. Описание отдельных аспектов распараллеливания в виде **независимых программных модулей** обеспечивается формированием совокупностей с **неопределенным типом**, который уточняется в других компонентах программы.

Использование языка Set@l открывает **принципиально новые возможности** для оперативной портации сложных программных комплексов на различные архитектуры ВС, в том числе **гибридные** и **реконфигурируемые**.

Благодарю за внимание!