Снежина Стоилова Фиданова

Софийски университет "Св. Климент Охридски"

Стефка Стоянова Фиданова

СИНТЕЗ НА СИСТОЛИЧЕСКИ МАСИВИ

ДИСЕРТАЦИЯ

за присъждане на образователната и научна степен
Доктор

НАУЧЕН РЪКОВОДИТЕЛ: проф. д.м.н. Райчо Лазаров
НАУЧЕН КОНСУЛТАНТ: ст.н.с. д-р Христо Джиджев

София, 1998

Глава 1

Увод

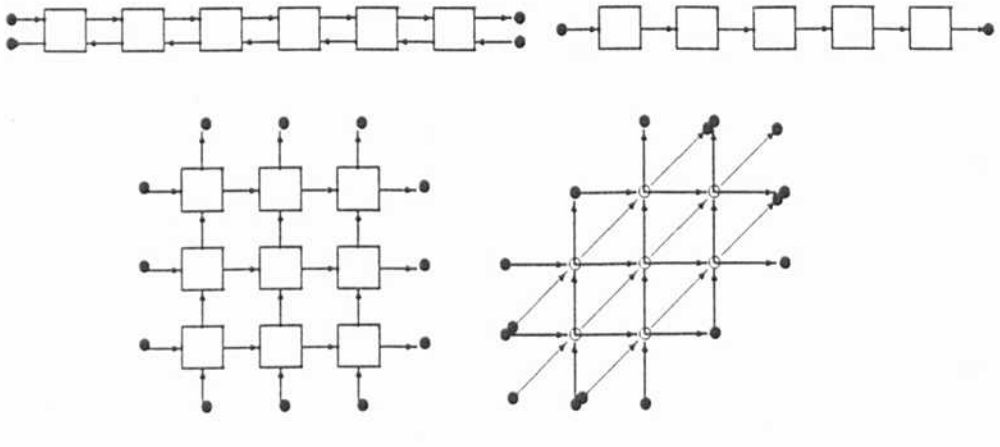
Развитието на силициевата технология през последните няколко десетилетия доведе до значително увеличаване на плътността и намаляване на цената на електронните устройства. Възникването на високо интегрирани системи (VLSI), съдържащи стотици хиляди транзистори оказва голямо въздействие върху архитектурата на съвременните изчислителни машини. Така става възможно да се реализират мощни компютърни структури върху един чип.

Паралелна архитектура е съвкупност от изчислителни елементи с връзки между тях, всеки от които може да има локална памет и/или всички да са свързани към обща памет.

Систолическите масиви, предложени от Kung и Leiserson [32] са интересна паралелна компютърна архитектура, много подходяща за директна реализация върху VLSI. Систолическият масив се състои от един или няколко вида изчислителни елементи, които ще бъдат наричани клетки, и изпълняват предварително фиксирани операции. Клетките извършват прости изчисления, имат АЛУ (аритметико-логическо устройство), евентуално локална памет. Клетките на систолическия масив работят синхронизирано. Под големина на масива се разбира броя на

изчислителните елементи от които е съставен. Между клетките съществуват локални връзки. В даден момент всяка клетка извършва някаква операция върху данните получени от съседните клетки и предава резултата. Систолическите масиви се отличават със своята проста и правилна структура и високата си специализация. Важна особеност на систолическите масиви е тясната връзка с алгоритъма изпълняван на масива. Предимствата на систолическите масиви водят до голям интерес към тях и постигане на интересни резултати.

На фигурите по-долу са показани различни видове систолически масиви.

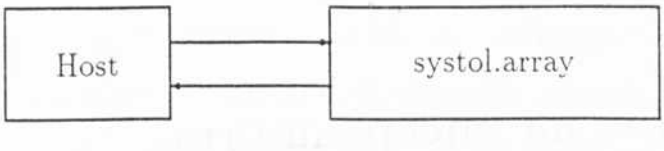


Фигура 1.1: Систолически масиви с различна архитектура

1.1 Актуалност на проблема

Идеята на Kung и Leiserson е систолическият масив да бъде периферно устройство на компютър с общо предназначение наречен (Host). Ролята на този компютър (Host) е да подава входните данни за решаваната

задача. Систолическият масив може да се използва за решаване както на една цяла задача, така и на разпаралелената част на една по-обща задача. Резултатът от паралелните изчисления се връща обратно на Host компютъра за получаване на крайното решение на задачата. Тази идея вече е реализирана дори в архитектурата на персонални компютри от последните поколения. Един от най-успешните продукти на германската фирма ISATEC, основана през 1989г. с цел разработване на паралелни компютри с ниска цена, е Systola 1024. Този продукт използва стандартен персонален компютър като Host. Systola 1024 се състои от 1024 процесора разположени като правоъгълна мрежа. Предназначен е за матрични изчисления и при направените изпитания е установено, че ускорява работата на персоналния компютър от 10 до 60 пъти.



Систолическите масиви са от категорията SIMD (Single Instruction Multiple Data), от класификацията на Flynn [26] за паралелни архитектури. Клетките на систолическия масив работят синхронизирано. За един такт всяка клетка получава данни от съседните клетки, извършва някакви операции и изпраща данни на своите съседи. Така се получава поток от данни преминаващ на тласъци (систоли) през систолическия масив, подобно на сърцето изтласкващо кръвта на равни тласъци. Оттам идва и името систолически.

Систолическият масив се състои от изчислителни елементи с фиксиран размер на локалната памет памет, ако има такава.

Чрез прилагане на нови технологии, цената на простите изчислителни елементи спада. Така цената на изчислителната система силно зависи от нейния дизайн. Чрез използването на регулярен и прост дизайн и използвайки VLSI технология може да се постигне намаляване на цената на архитектурата.

Важен фактор за постигане на висока скорост на изчислителната система е паралелизмът. При системите със специално предназначение, архитектурата зависи от алгоритъма, който се изпълнява.

Систолическият масив приема, данните от Host компютър, а след това му връща резултата. Намалява се броят на входно/изходните операции и обръщенията към външна памет.

Посочените свойства са основа за използване на систолическите масиви, при задачи с еднотипни изчисления върху голям масив от данни.

Гореизложеното обосновава актуалността на дисертацията.

1.2 Цел на дисертацията

Цел на дисертационния труд е изследването на систолически масиви, създаване на алгоритми за решаване на широко използвани в практиката задачи и разполагането им върху систолически масиви.

В съответствие с тази цел основните задачи в дисертацията са:

1. Изследване на задачи от типа на задачата за раницата:

- Реализация на задачата върху линеен систолически масив;
- Реализация на задачата върху двумерен систолически масив.

2. Разширяване на систолически масиви:

- Разширяване на систолически масив, съдържащ само един тип изчислителни елементи;
- Разширяване на систолически масив, съдържащ няколко типа изчислителни елементи.

1.3 Обосновка и състояние на проблема

Обикновено систолическите масиви са специализирани за изпълнението на конкретен алгоритъм. Цели се постигане на максимален паралелизъм с минимум комуникации. Вземайки предвид модулярността, регулярността и локалните връзки, систолическите масиви са удобни за VLSI реализация.

Систолическите масиви са много удобни за решаването на някои задачи от линейната алгебра (конволюция, трансформация на Фурие, матрични изчисления приложими за решаване на линейни системи) [34, 40, 43, 47, 48]. Със систолически масиви могат да се решават не само числени, но и други сложни задачи като разшифроване на кодове, обработка на биологична информация и др. [2, 24, 56].

Едни от методите за синтез на систолически масиви са за решаване на хомогенни рекурентни уравнения [52, 53]:

$$u(i,j) = F(u(i-p,j), v(i,j-q)) \quad (i,j) \in D \; p,q = const.,$$

където $D-$ областта в която се изменят променливите на уравнението $u, v-$ функции, F- функция.

Едно от направленията за изследване на систолическите масиви е свързано със задачата за създаване на методология за автоматична

трансформация на алгоритмите върху систолически масиви, които да реализират тези алгоритми. Методите за проектиране на паралелни архитектури, в частност методите за проектиране на систолически масиви съдържат изобразяване на множеството от операции извършвани от алгоритъма върху пространство/времева графика, описваща извършването на изчисленията във времето. Това изобразяване показва всеки елемент от множеството на изчисленията в кой изчислителен елемент ще бъде реализиран и в кой момент време.

Предмет на дисертационният труд е изследването на систолическите масиви. Разработени са алгоритми за решаване на широко използвани в практиката задачи и разполагането им върху систолически масиви. За описание на алгоритмите и на хардуера се използват графи [11]. Ще използваме означенията и понятията въведени от Хр. Джиджев [19]

Ориентиран граф G се нарича наредена двойка (V, E) от множества, където елементите на V се наричат *върхове* на G, а елементите на E са наредени двойки от върхове и се наричат *ребра* на G, т.е. E се съдържа в $V \times V$.

В дисертацията ще се използват само ориентирани графи и за кратко ще бъдат наричани само графи.

Ако $e = (v, w) \in E$, то v се нарича *начало*, а w - *край*. За върховете v и w ще се казва, че са *съседни*, а реброто e е *инцидентно* с тях. Броят на инцидентните с даден връх ребра се нарича *степен* на върха.

Път се нарича такава последователност от върхове $p = (v_0, v_1, \ldots, v_k)$, за която $(v_{i-1}, v_i) \in E$, $i = 1, ..., k$, числото k се нарича *дължина* на пътя. Ако $v_0 = v_k$, то p е *цикъл*.

За представяне на алгоритмите и систолическите масиви ще използваме графи с етикети и цветове на ребрата и цветове на върховете.

Поточен граф се нарича ацикличен граф с цветове на върховете и

цветове и етикети на ребрата, удовлетворяващ условията:

1. Ребра с еднакъв етикет имат еднакъв цвят.

2. За всеки етикет s съществува път започващ и завършващ във върхове от степен 1 и съдържащ всички ребра с етикет s и само тях.

3. За всеки цвят c и за всеки връх v съществува най-много едно ребро с цвят c влизащо/излизащо от v.

4. За всяка двойка от върхове (v, w), всички пътища между v и w имат еднаква дължина.

Нека G е поточен граф. Ще въведем релация на частична наредба " $\leq$ ". За върховете $v, w \in G$ казваме, че $v \leq w$, ако съществува път от v до w. Минималните върхове по отношение на тази релация се наричат *входни*, а максималните - *изходни*. Всички входни и изходни върхове образуват множеството на *терминалните* върхове. Ребрата инцидентни с тях се наричат *терминални ребра* (входни и изходни).

Ще използваме следните означения, ако $v \in V(G)$ и $e = (v, w) \in E(G)$, то с $col(v)$ ще означаваме цвят на v, с $col(e)$ - цвят на e, а с $lab(e) = lab((v, w))$ - етикет на e.

Нека имаме алгоритъм за решаване на дадена задача. Ще построим граф $G = (V, E)$, по следния начин. На всяка операция от алгоритъма ще съпоставим връх от графа. Ако аргументът на една операция е получен, като резултат от изпълнението на друга операция, то съответните върхове ще свържем с ребро, започващо от върха откъдето се взема резултата. На входните и изходните данни съответстват терминални върхове. Ако аргументът на резултата е начална стойност или резултатът от операцията не се използва (т. е. изходни данни), то съответното ребро е терминално. Граф $G = (V, E)$ ще наричаме *граф на зависимостите*.

За изобразяване на систолическите масиви също ще използваме гра-

фи. Върховете ще съответстват на изчислителните елементи, а ребрата - на физическите връзки между тях. Този граф ще наричаме хардуерен граф.

Алокираща функция $alloc : G \to H$ ще наричаме изображение на графа на зависимостите G, за даден алгоритъм, върху хардуерния граф H изобразяващ даден систолически масив. При това изображение на всеки връх от граф G съответства точно един връх от H и в един връх от H могат да бъдат изобразени повече от един връх на G.

Реализацията на алгоритъма, изобразен чрез граф на зависимостите G, върху систолически масив, изобразен чрез граф H, ще разделим на времеви стъпки (моменти). В един момент в клетките на систолическия масив, изобразени чрез върховете на граф H, се извършват изчисления и предаване на данни към съседните клетки. Броят на моментите (стъпките) за получаване на резултата ще наричаме *време за реализация* на алгоритъма, върху систолическия масив (*времева сложност*).

Времева функция ще наричаме функция $t : V(G) \to N$, където $V(G)$ е множеството от върховете на G, а N е множеството на целите положителни числа.

Времева функция отговаряща на условията:

$1.t(v) \neq t(w)$, ако $alloc(v) = alloc(w)$, $v, w \in V(G)$;

$2.t(v) < t(w)$, ако от връх v излиза ребро, влизащо във връх w, $v, w \in V(G)$;

ще наричаме *валидна времева функция.*

Когато поточен граф (граф с цветове на върховете и цветове и етикети на ребрата) се използва за описание на даден алгоритъм (граф на зависимостите), върховете му съответстват на изчисленията, а ребрата на зависимостта между тях. Цветовете на ребрата съответстват на различните потоци данни, а етикетите на придвижването на една инфор-

мационна единица. Цветовете на върховете съответстват на различните видове изчисления.

Когато поточният граф се използва за изобразяване на систолически масив (хардуерен граф), върховете съответстват на изчислителните елементи, а ребрата на физическата връзка между тях. Цветовете и етикетите на ребрата показват придвижването на информацията, а цветовете на върховете съответстват на различните видове изчислителни елементи.

Всеки поточен граф може да се разглежда като хардуерен граф или граф на зависимостите в зависимост от интерпретацията.

Преминаването на данните през клетките на систолическия масив води до проблеми като активиране на клетките и синхронизация на въвеждането на данните през крайните клетки на систолическия масив. Решаването на тези проблеми е важно за реализацията на дадения алгоритъм върху систолическия масив.

Проблемът за активиране на клетките идва от преминаването на данните през всички клетки на масива. Въпросът е кои данни да се считат за "анонимни"; отделната клетка не знае мястото си в масива, нито кои данни се намират в паметта и. За това успоредно с данните се придвижва информация за контрол на масива, която показва дали клетката трябва да извършва изчисления или да е неактивна.

Проблемът за подреждане на данните е проблем за вход на данните, които не са записани предварително в локалната памет и е нужно да бъдат подредени по определен начин и подавани на систолическия масив (от Host компютъра) в определени моменти.

Дисертацията се състои от увод, оформен като първа глава и още три глави.

Във глава втора е разгледан алгоритъм за решаване на нехомогенни рекурентни уравнения за задачи от типа на задачата за раницата и разполагането му върху линеен систолически масив.

Можем да формулираме задачата за раницата по следния начин: имаме раница с капацитет c, в която можем да сложим m типа предмети. Всеки предмет от тип i има тегло $f(i)$ и цена $g(i)$, като $f(i)$, $g(i)$, m и c са цели положителни числа. Трябва от предмет от тип i да поставим в раницата z_i броя така, че цената на предметите в нея да е максимална, без да надхвърляме капацитета и, т.е.:

$$\max\left\{\sum_{i=1}^{m} g(i)z_i : \sum_{i=1}^{m} f(i)z_i \leq c,\ z_i \geq 0 \text{ цяло число},\ i = 1,\ldots,m\right\}.$$

От този тип са задачата за раницата без ограничения (UKP), задачата за сумиране на подмножества (SSP), задача за рестото (CMP) и др. Задачите от типа на задачата за раницата имат голямо приложение в икономиката и производството, например при подреждане на товари в кораби, разкрояване, контролиране на бюджет. Тези задачи водят до следния ограничен клас от нехомогенни рекурентни уравнения:

$$u[i,j] = F(u[i-1,j], u[i-l, j-f(i)] + g(i)).$$

Уравнението е дефинирано за $0 < i \leq m$, $0 < j \leq c$ и следните гранични условия: $u[i,0] = 0$, $u[i,j] = 0$ при $j < 0$. Като $l \in \{0,1\}$ и $u[0,j]$ е дадено. Целочислените функции $f(i)$ и $g(i)$ са известни предварително за $i = 1,2,\ldots,m$ и стойностите $u[m,j]$ трябва да бъдат изчислени. До горното уравнение водят следните задачи:

- UKP Ако F е max, $l = 0$, $u[0,j] = 0$ за $0 \leq j \leq c$ $1 \leq i \leq m$;

- CMP Ако F е min, $l = 0$, $g(i) = 1$, $u[0,0] = 0$, $u[0,j] = \infty$ за $1 \leq j \leq c$ $1 \leq i \leq m$;

- SSP Ако F е max, $l = 1$, $f(i) = g(i)$, $u[0,j] = 0$ за $0 \leq j \leq c$ $1 \leq i \leq m$.

Стремежът в тази глава е да се създаде линеен систолически масиви за горепосочената задача. Всяка клетка на систолическия масив има локална памет с фиксиран размер α регистъра, използвани за запазване на междинните изчисления. Числото α не зависи от параметрите на задачата.

Съществуват различни паралелни реализации на задачата за раницата. Andonov и Benaini [3] са едни от първите, които разглеждат реализация на задачата за раницата върху систолически масив. В случая, когато няма ограничение за броя на използваните клетки, тяхната времева сложност е $mf_{max} + c$ върху mf_{max} изчислителни елемента с фиксирана памет 1 регистър, $f_{max} = \max_i f(i)$. Недостатък на тяхната реализация е, че имат изчислителни елементи, които не извършват изчисления, а само предават данни. Lin и Storer [39] предлагат реализация на тази задача върху хиперкуб с p процесора с времева сложност $O((\frac{mc}{p} + c^2)log(p))$, но локалната памет на изчислителните елементи зависи от параметрите на задачата. Chen, Chern и Jang [13] описват линейна архитектура с p процесора с времева сложност $O\left(\frac{mc}{p} + m\right)$, но локалната памет на изчислителните елементи зависи от параметрите на задачата. Броят на клетките при някои реализации е твърде голям. Например алгоритъмът на Teng [58] изисква $M(c)$ изчислителни елемента и има времева сложност $O(log^2(mc))$. Където $M(n)$ е броят на клетките, необходими за умножение на две $n \times n$ матрици за време $O(log(n))$ (това число е между n^2 и n^3). В [5] Andonov и Quinton постигат времева сложност $O\left(\frac{mcf_{max}}{p} + p\right)$ върху p процесора. Това е обобще-

ние на реализацията в [3], но когато броят на клетките е фиксиран. В [4] Andonov и Gruau реализират задачата върху двумерен правоъгълен масив, състоящ се от p изчислителни елемента, с времева сложност $O\left(\frac{mc}{p}\right)$, но при тях локалната памет на отделната клетка е достатъчно голяма, т.е. зависи от параметрите на задачата.

Целта е да се намерят съответно времева и алокираща функции. Алокиращата функция изобразява няколко върха от графа на зависимостите в един и същи връх на хардуерния граф, чрез който изобразяваме линеен систолически масив. Неотрицателната времева функция показва в кой момент време се извършва дадено изчисление. Тя трябва да бъде такава, че данните необходими за дадено изчисление да са получени преди извършване на изчислението.

Ще предполагаме, че разполагаме с толкова изчислителни елемента, колкото са ни необходими. Конструирането на линеен систолически масив може да се раздели на два етапа:

1. Директно изобразяване на графа на зависимостите върху линеен хардуерен граф състоящ се от m нетерминални върха. Така локалната памет на клетка с номер i от линейния систолически масив трябва да има $f(i)$ регистъра, т.е. зависи от параметрите на задачата. Такъв систолически масив се нарича *немодулярен*.

2. За да бъде получен *модулярен* систолически масив (т.е. локалната памет да не зависи от параметрите на задачата) с фиксирана памет α регистъра, клетка с номер i се заменя с $\left\lceil \frac{f(i)}{\alpha} \right\rceil$ клетки ако $f(i) \leq \frac{c}{2}$ и с $\left\lceil \frac{c-f(i)+1}{\alpha} \right\rceil$, ако $f(i) > \frac{c}{2}$, по дължина на масива. С w_i ще бележим броя на клетките заменили клетка i.

Така алокиращата функция ще бъде:

$$alloc(i,j) = \sum_{s=1}^{i-1} w_s + k,$$

$$\text{където } k = \begin{cases} \left\lceil \frac{c-f(i)+1}{\alpha} \right\rceil & \text{за } f(i) > \frac{c}{2},\ f(i) \geq j > c - f(i) \\ \left\lceil \frac{j \bmod f(i)+1}{\alpha} \right\rceil & \text{в останалите случаи} \end{cases},$$

(i, j) е връх от графа на зависимостите, а дясната страна на равенството е номер на връх от хардуерния граф.

Разглеждаме следната времева функция:

$$t(i,j) = \begin{cases} j + k & i = 1 \\ \sum_{s=1}^{i-1} w_s + j + k & 2 \leq i \leq m \\ \sum_{s=1}^{i-1} w_s + j & i = m+1 \end{cases},$$

$$\text{където } k = \begin{cases} \left\lceil \frac{c-f(i)+1}{\alpha} \right\rceil & \text{за } f(i) > \frac{c}{2},\ f(i) \geq j > c - f(i) \\ \left\lceil \frac{j \bmod f(i)+1}{\alpha} \right\rceil & \text{в останалите случаи} \end{cases}.$$

Тогава времевата сложност на алгоритъма ще бъде:

$T = \sum_{i=1}^{m} w_i + c \leq \sum_{i=1}^{m} \left\lceil \frac{f(i)}{\alpha} \right\rceil + c$ върху $\sum_{i=1}^{m} w_i$ процесора.

Описали сме функционирането на отделната клетка на систолическия масив и контролните символи, които са необходими за правилното и функциониране. Използваме един управляващ символ, който показва съответно след колко клетки се намира клетката, със съответните данни, в която ще се извършват изчисления. В случая, когато $\alpha = 1$ и всички стойности на $f(i)$ са равни, за $i = 1, \ldots, m$, нашият резултат се изравнява с резултата на Andonov и Benaini [3]. В останалите случаи получаваме по-добър резултат както за броя на изчислителните елементи, така и за времевата сложност на реализацията.

Разгледан е и случаят когато линейният масив е с фиксиран брой

изчислителни елементи, p. Тогава времевата сложност на алгоритъма е:

$$t = c\left\lceil \frac{\sum_{i=1}^{m} w_i}{p} \right\rceil + p \leq c\frac{\sum_{i=1}^{m} \left\lceil \frac{f(i)}{\alpha} \right\rceil}{p} + p.$$

Когато всички стойности на $f(i)$ са равни, $i = 1, \ldots, m$, и $\alpha = 1$, нашият резултат се изравнява с резултата получен от Andonov и Quinton [5]. В останалите случаи получаваме по-добър резултат.

В трета глава предлагаме реализация на горе описаното нехомогенно рекурентно уравнение върху двумерен правоъгълен систолически масив. Двумерният правоъгълен систолически масив, състоящ се от $n \times q$ изчислителни елемента, може да се представи, чрез граф $G(V, E)$. На всеки връх от графа се съпоставя наредена двойка числа (i, j), $i = 1, \ldots, n$, $j = 1, \ldots, q$. Между два върха $v = (i, j)$ и $w = (i', j')$ има ребро само, ако $|i - i'| + |j - j'| = 1$. Подобно на предходния случай, локалната памет на отделната клетка на двумерния правоъгълен систолически масив е с фиксиран размер α регистъра, независещ от параметрите на задачата, предназначени за запазване на междинните изчисления. Предполагаме, че разполагаме с толкова изчислителни елемента, колкото са ни необходими. Конструирането на двумерен правоъгълен систолически масив може да се раздели на два етапа:

1. Директно изобразяване на графа на зависимостите върху линеен хардуерен граф, състоящ се от m нетерминални върха. Така ни е необходима локална памет $f(i)$ регистъра в клетка i. Това означава, че паметта зависи от параметрите на задачата и получаваме немодулярен масив.

2. За да получим модулярен систолически масив с фиксиран раз-

мер на локалната памет α регистъра, заменяме клетка i с $\lceil \frac{f_{max}}{\alpha} \rceil$ клетки. $f_{max} = max_i f(i)$, така че първата клетка от клетките заменящи клетка i се свързва с първата клетка от клетките заменящи клетка $i+1$, съответно втората клетка от клетките заменящи клетка i се свързва с втората клетка от клетките заменящи клетка $i+1$, $i = 1, \ldots, m-1$. Така получаваме двумерен правоъгълен систолически масив ,а който всяка клетка има фиксирана памет α регистъра.

В този случай алокиращата функция ще бъде:

$$alloc(i,j) = (i,k),$$

където $k = \begin{cases} jmod \lceil \frac{f(i)}{\alpha} \rceil & jmod \lceil \frac{f(i)}{\alpha} \rceil \neq 0 \\ \lceil \frac{f(i)}{\alpha} \rceil & jmod \lceil \frac{f(i)}{\alpha} \rceil = 0 \end{cases}, \quad i = 1, \ldots, m, \quad j = 1, \ldots, c.$

Разглеждаме следната времева функция:

$$t(i,j) = (i-1) \left\lceil \frac{f_{max}}{\alpha} \right\rceil + \left\lceil \frac{j}{\lceil \frac{f_{min}}{\alpha} \rceil} \right\rceil .$$

Получаваме, че времевата сложност на алгоритъма е:

$$T = m \left\lceil \frac{f_{max}}{\alpha} \right\rceil + \left\lceil \frac{c}{\lceil \frac{f_{min}}{\alpha} \rceil} \right\rceil$$

върху $p = m \times \lceil \frac{f_{max}}{\alpha} \rceil$ изчислителни елемента.

Описали сме функционирането на отделната клетка на систолическия масив и контролните символи, които са необходими за правилното и функциониране. Използваните управляващи символи са като в случая на линеен систолически масив.

Разглеждаме и случая когато двумерният правоъгълен систолически масив е с фиксиран брой, $p = n \times q$, изчислителни елемента. Получихме, че времевата сложност на цялата задача е:

$$T = \left\lceil \frac{m}{n} \right\rceil \left\lceil \frac{\left\lceil \frac{f_{max}}{\alpha} \right\rceil}{q} \right\rceil \left\lceil \frac{c}{min\{q, \left\lceil \frac{f_{min}}{\alpha} \right\rceil\}} \right\rceil + nq$$

Нека разгледаме случая, когато $\alpha = 1$, n дели m, q дели f_{max}. Тогава получаваме, че времевата сложност на алгоритъма реализиран върху двумерен правоъгълен систолически масив е:

$$T = \frac{mcf_{max}}{nq \min\{q, f_{min}\}} + nq$$

Когато $q = 1$ или $f_{min} = 1$ нашият резултат се изравнява с резултата на Andonov и Quintonв [5], в останалите случаи нашият резултат е по-добър.

Нека с T_1 бележим времевата сложност на алгоритъма за решаване на задачи от типа на задачата за раницата върху линеен систолически масив, а с T_2 отбележим времевата сложност на задачата, но върху двумерен правоъгълен систолически масив. Нека и двата систолически масива бъдат с еднакъв брой изчислителни елементи. Коя от двете реализации е по-добра зависи от това, коя от стойностите $\sum_{i=1}^{m} f(i)$ и $\frac{mf_{max}}{\min\{q, \frac{f_{min}}{\alpha}\}}$ е по-малка. Тогава при стойности на $f(i)$ близки до f_{min}, $i = 1, \ldots, m$ имаме $T_1 < T_2$. Обратно, ако стойностите на $f(i)$ са близки до стойностите на f_{max} то $T_1 > T_2$. Когато $q = 1$, т.е. двумерният масив стане линеен $T_1 \leq T_2$.

Така, ако разполагаме с някаква архитектура, например хиперкуб, можем да преценим, как е по-добре да реализираме задачата. В едни случаи е по-добре да използваме реализацията върху линеен систоли-

чески масив, а в други случаи по-подходящо е да използваме реализацията върху двумерен правоъгълен систолически масив.

В предишните глави показахме реализацията на един конкретен алгоритъм върху линеен и двумерен правоъгълен систолически масиви. До тук ставаше дума за един сравнително прост масив, съдържащ само един тип изчислителни елементи. При по-сложни систолически масиви и особено когато съдържат повече от един тип изчислителни елементи, задачата за реализацията на отделен алгоритъм става изключително трудна. Поради това много актуален е въпросът за автоматизация на процеса на проектиране.

Изследователите използват своя опит и интуиция за конструиране на различни систолически масиви, докато въпросът за тяхната оптималност и коректност се изследва след това. Друго направление свързано със задачата за конструиране на систолически масиви е да се открие методология за автоматично трансформиране на алгоритмите върху систолически масиви, които да реализират тези алгоритми коректно. Тази задача се решава по различен начин; трансформации върху системи рекурентни уравнения [18, 27, 46] и оптимизиране [41], геометрични трансформации [33] и др.

В глава четвърта разглеждаме метод за синтез на систолически масиви чрез тяхното разширяване. Този подход е предложен от Хр. Джиджев [19, 18], но там са разгледани само систолически масиви с правоъгълна структура, състоящи се от един тип изчислителни елементи.

Нашият подход се основава на идеята да изследваме директно не алгоритъма и изчислителната архитектура, а структури, които са значително по-малки и все пак запазват основните свойства на изходната архитектура. Малкият размер на тези структури прави възможно раз-

глеждането на голям брой кандидати за изобразяване. Така можем да намерим най-доброто решение по някакъв критерий.

Първо сме разгледали *хомогенни* систолически масиви, т.е. систолически масиви които можем да представим чрез хардуерни графи с върхове оцветени в един и същ цвят. Това съответства на алгоритми при които се извършват само един тип операции. Основно понятие при този метод са рекурсивните редици от графи.

Нека е даден един поточен граф G_0. Всеки връх от този граф ще заменим със същия граф, като ребрата свързващи два съседни върха в G_0 ще заменим с пътища от същият цвят. Върховете по тези пътища ще се наричат *допълнителни*. Доказали сме, че допълнителните върхове поставени за ребра от даден цвят, са допълнителни и за ребрата от останалите цветове. По този начин получаваме граф G_1, който ще наричаме *разширение* на G_0 с G_0. Ако всеки връх на G_1 заменим с граф G_0 получаваме граф G_2, който е разширение на G_1 с G_0 и т.н. получаваме редица от графи породена от граф G_0, която ще наричаме *рекурсивна редица*. Поточен граф, за който можем да построим рекурсивна редица ще наричаме разширим поточен граф. Намерено е необходимо и достатъчно условие граф G да бъде разширим поточен граф. Доказали сме, че ако $G_0, G_1, \ldots$ е рекурсивна редица от графи, породена от поточния граф G_0, то за всяко $i > 0$, G_i също е поточен граф.

Нека $G_0, G_1, \ldots$ и $H_0, H_1, \ldots$ са съответно рекурсивни редици от графи на зависимостите и хардуерни графи, като G_j е граф на зависимостите съответстващ на хардуерния граф H_j. $M_0 = (S_0, T_0)$ е изображение съставено от две изображения, S_0 е изображение на граф G_0 върху граф H_0, а T_0 е времева функция. Рекурсивно дефинираме редица от изображения $M_0, M_1, \ldots, M_j = (S_j, T_j)$. Доказали сме, че S_j е изображение на граф G_j върху граф H_j, а T_j е времева функция.

Разгледали сме и *хетерогенни* систолически масиви, т.е. систолически масиви състоящи се от няколко вида изчислителни елементи. Това означава, че както в програмния така и в хардуерния граф върховете са оцветени с повече от един цвят или това съответства на алгоритми, при които се извършват няколко типа операции. До сега подобен проблем е разглеждан само за хомогенни систолически масиви.

Хетерогенен поточен граф се разширява, като от него получим хомогенни графи. Нека G е поточен хетерогенен граф и нека върховете на G бъдат оцветени в s цвята. От граф G ще получим q на брой графа $(q \geq s)$ $D_1, \ldots, D_q$ по следния начин:

1. D_i, $i \in [1, q]$ съдържа нетерминални върхове само от един и същи цвят и свързващите ги ребра;

2. $\cup_{i=1}^{q} V_n(D_i) = V_n(G) \quad \cup_{i=1}^{q} E_n(D_n) \subset E_n(G)$

3. Ако $x \in V_n(D_i)$, $y \in V_n(D_j)$, $i \neq j$ и $(x, y) \in E_n(G)$ то в граф D_i от върха x излиза терминално ребро с цвят равен на $col(x, y)$, а в граф D_j във върха y влиза терминално ребро от същият цвят.

4. Ако $x \in V_n(D_j)$, $y \in V_n(D_i)$, $(x, y) \in E_n(G)$ и $col(x, y) = c$, тогава графите D_i и D_j имат равен брой терминални ребра от цвят c.

Доказали сме, че $D_1, \ldots, D_q$ са поточни графи. Ако $D_1, \ldots, D_q$ са разширими, то те се разширяват като хомогенни поточни графи, след което разширенията им отново се свързват. Така се получава разширение на хетерогенния граф G. Аналогично на хомогенния случай се построява рекурсивна редица от хетерогенни графи, породени от граф G. Доказано е, че необходимо и достатъчно условие поточният граф G да е разширим е хомогенните графи $D_1, \ldots D_q$ да бъдат разширими. Разгледана е коректността на алгоритъма за разширяване на хетерогенни поточни графи.

Нека е дадена рекурсивна редица породена от хетерогенния поточен

граф G_0. Тогава всички графи от редицата също са хетерогенни поточни графи. Ако графите, на които се разделя хетерогенния поточен граф се състоят от един връх, то разширението на графа съвпада със самия граф.

Аналогично на хомогенния случай сме разгледали рекурсивни редици от графи на зависимостите и хардуерни графи и сме построили рекурсивна редица от изображения, така че i-тият член на редицата от изображения изобразява i-тият член на редицата от графи на зависимостите върху i-тият член на редицата от хардуерни графи.

1.4 Приноси на дисертацията

Резултатите в дисертационния труд и приносите в него могат да бъдат обобщени както следва:

- Конструирани са линеен и двумерен правоъгълен систолически масиви за задачата за раницата и подобни на нея задачи. Клетките на тези масиви имат локална памет с фиксиран размер, независещ от параметрите на задачата. В случая когато няма ограничение за броя на използваните клетки е подобрена времевата сложност и броя на изчислителните елементи. Когато броят на клетките е фиксиран е получена по-добра времева сложност от резултатите на други автори.

- Обобщен е алгоритъмът на Хр. Джиджев за разширяване на хомогенни систолически масиви. Новият алгоритъм обхваща по-голям клас систолически масиви. Доказана е коректността на алгоритъма. Намерено е необходимо и достатъчно условие един хомогенен

систолически масив да е разширим.

- Предложен е алгоритъм за разширяване на хетерогенни систолически масиви. Доказана е коректността на алгоритъма. Намерено е необходимо и достатъчно условие един хетерогенен систолически масив да бъде разширим.

1.5 Метод на изследването

При изследването на алгоритмите и систолическите масиви използваме означения и понятия от теорията на графите. За описание на изображението на алгоритъма върху систолическия масив и получаване на алокираща функция прилагаме метода на покритията. При разширяването на хомогенни систолически масиви (състоят се от един тип изчислителни елементи), използваме замяната на един връх от графа описващ масива с граф, докато при хетерогенните систолически масиви (състоят се от повече от един тип изчислителни елементи) използваме разделяне на графа на хомогенни части.

1.6 Апробация

Части от дисертационната работа са докладвани на международни конференции и на семинара по Паралелни алгоритми в ЦЛПОИ - БАН. Някои от работите са използвани в договорите МУ02/92, МУ20/94, МУ02/96 с Националния фонд **Научни изследвания**. Част от резул-

татите са цитирани и използвани в книгата на G.M. Megson "Parallel Algorithms for Knapsack Type Problem", World Scientific Publ,1998.

1.7 Публикации във връзка с дисертацията

Основните резултати от дисертационния труд са публикувани в 9 статии, една от които в сп.Parallel Algorithms and Aplications, една в Университета в Нюкасъл, 5 от тях са публикувани в чужбина като издания на трудове на международни конференции от Nord Holand, World Scientific Pub., IOS-Pres, POLMET-Press, Akademie Verlag и две публикувани у нас съответно от Академичното издателство и от издателство DATECS.

1.8 Структура и обем на дисертацията

Дисертационният труд се състои от увод, съдържащ обосновка на проведеното изследване, три глави, заключение и списък на използваната литература. Текстът съдържа 106 страници и включва 17 фигури.

Глава 2

Линейна реализация на задачи от типа на задачата за раницата

Задачата за раницата може да се формулира по следния начин: имаме раница с капацитет c, в която можем да сложим m типа предмети. Всеки предмет от тип i има тегло $f(i)$ и цена $g(i)$, ($f(i)$, $g(i)$, m и c са цели положителни числа). Трябва от предметите от тип i да поставим в раницата z_i броя така, че цената на предметите в нея да е максимална, без да надхвърляме капацитета и, т.е.:

$$\max\left\{\sum_{i=1}^{m} g(i)z_i : \sum_{i=1}^{m} f(i)z_i \leq c,\ z_i \geq 0 \ \text{ цяло число}, i = 1, 2, \ldots, m\right\}.$$

Един от най известните методи за решаване на задачата за раницата е методът на динамичното програмиране. Той се базира на принципа за оптималност и обикновено съдържа два етапа. На първия етап се изчислява оптималната стойност на ценовата функция. На втория етап тя се използва за конструиране на вектора $z = (z_1, \ldots, z_m)$. Дефинираме функция $u[i, j]$, която показва оптималното решение когато се избира

само от първите j предмета и капацитетът на раницата е равен на i.

Задачата се свежда до следното рекурентно уравнение :

$$u[i,j] = F(u[i-1,j], u[i-l, j-f(i)] + g(i)). \tag{2.1}$$

Уравнението е дефинирано за $0 < i \leq m$, $0 < j \leq c$ и следните гранични условия: $u[i,0] = 0$, $u[0,j] = 0$ при $j < 0$, $u[0,j]$ е дадено. При задачата за раницата без ограничения $F = \max$, а $l = 0$. Други задачи водещи до уравнение (2.1) са 0/1 задача за раницата (0/1KP) [44], задача за рестото (CMP) [44], задача за сумиране на подмножества (SSP) [44], задача за най-дълга обща подредица (LCS) [14].

- UKP Ако F е max, $l = 0$, $u[0,j] = 0$ за $0 \leq j \leq c$;
- CMP Ако F е min, $l = 0$, $g(i) = 1$, $u[0,0] = 0$, $u[0,j] = \infty$ за $1 \leq j \leq c$;
- SSP Ако F е max, $l = 1$, $f(i) = g(i)$ $u[0,j] = 0$ за $0 \leq j \leq c$;
- 0/1KP Ако F е max, $l = 1$, $u[0,j] = 0$ за $0 \leq j \leq c$;

Тези задачи имат голямо приложение [28, 29, 39, 44].

В тази глава ще разглеждаме уравнение (2.1) в случаите $l = 0$ и $l = 1$.

Нашата цел е да конструираме линеен систолически масив на който да реализираме уравнение (2.1). Този масив ще се състои от изчислителни елементи (наричани клетки), имащи локална памет с размер α регистъра, използвана за запазване на междинните резултати. Числото α не зависи от параметрите на задачата.

Съществуват различни реализации на задачата за раницата базирани на метода на Динамичното програмиране [3, 5, 7, 8, 9, 10, 23, 13,

39, 42]. Andonov и Benaini [3] са едни от първите, които разглеждат реализация на задачата за раницата върху систолически масиви. Те разглеждат случая, когато разполагат с толкова изчислителни елементи, колкото са им необходими. При тях времевата сложност е $mf_{max}+c$ върху mf_{max} процесора, $f_{max} = \max_i f(i)$. Недостатък на тази реализация е, че има изчислителни елементи, които не извършват операции, а само предават данни. Lin и Storer [42] предлагат алгоритъм за $0/1KP$ върху хиперкуб с p изчислителни елемента с времева сложност $O((\frac{mc}{p}+c^2)log(p))$, но локалната памет на изчислителните елементи зависи от параметрите на задачата. Chen, Chern и Jong [13] предлагат линейна немодулярна архитектура с p изчислителни елемента с времева сложност $O(\frac{mc}{p}+m)$, но локалната памет зависи от броя на изчислителните елементи. Броят на клетките елементи при някои реализации е твърде голям. Например реализацията на Teng [58] изисква $M(c)$ клетки и има времева сложност $O(log^2(mc))$, $M(n)$ означава броя на изчислителните елементи необходими за умножение на две $n \times n$ матрици за време $O(log(n))$ (това число е между n^2 и n^3). В [5] Andonov и Quinton постигат времева сложност $O\left(\frac{mcf_{max}}{p}+p\right)$ върху p изчислителни елемента. Това е обобщение на реализацията в [3], но когато разполагаме с фиксиран брой изчислителни елементи. В [5] Andonov и Gruau реализират задачата върху двумерен правоъгълен систолически масив, състоящ се от p изчислителни елемента с времева сложност $O(\frac{mc}{p})$, но при тях локалната памет на отделната клетка е достатъчно голяма, т.е. зависи от параметрите на задачата.

2.1 Формулировка на задачата

Паралелизацията на много алгоритми може да бъде получена чрез използването на трансформации, които се прилагат върху рекурентни уравнения. За първи път те са използвани от Karp [36] през 1967г. Този метод е приложен от Lamport [37] за паралелизация на цикли и е основа за редица разработки върху синтеза на систолически масиви [12, 35, 46, 52].

Дефиниция 1 *Алокираща функция* $alloc : G \to H$ *ще наричаме изображение на графа на зависимостите* G, *за даден алгоритъм , върху систолическия масив изобразен чрез граф* H. *При това изображение на всеки връх от граф* G *съответства точно един връх от граф* H. *В един връх от граф* H *могат да бъдат изобразени повече от един връх на граф* G.

Реализацията на алгоритъма, изобразен чрез граф на зависимостите G, върху систолически масив, изобразен чрез граф H, ще разделим на времеви стъпки (*моменти*). В един момент в изчислителните елементи на систолическия масив, изобразен чрез граф H, се извършват изчисления и предаване на данни към съседните клетки. Броят на моментите, за получаване на резултата, ще наричаме време за реализация на алгоритъма върху систолическия масив (*времева сложност*).

Дефиниция 2 *Времева функция ще наричаме функцията* $t : V(G) \to N$, *където* $V(G)$ *е множеството от върховете на графа на зависимостите* G, *а* N *е множеството от целите неотрицателни числа.*

Дефиниция 3 *Времева функция* t *отговаряща на условията:*

1.$t(v) \neq t(w)$ *ако* $alloc(v) = alloc(w), \quad v, w \in V(G)$;

2. $t(v) < t(w)$, *ако от връх* v *излиза ребро, влизащо във връх* w, $v, w \in V(G)$;

ще бъде наричана валидна времева функция.

Алгоритъмът за решаване на задачата, описана чрез уравнение (2.1), ще представим чрез граф на зависимостите, $G = (V, E)$ с крайно множество от върхове V и крайно множество от ребра E. Върховете изобразяват извършваните операции, а ребрата - зависимостите между тях.

Ще опишем модулярен систолически масив. Това означава, че той съдържа еднакви клетки с фиксиран размер на локалната памет α регистъра на всяка от клетките, независещ от параметрите на задачата. За тази цел ще използваме граф, на който върховете съответстват на изчислителните елементи, а ребрата - на връзките между тях.

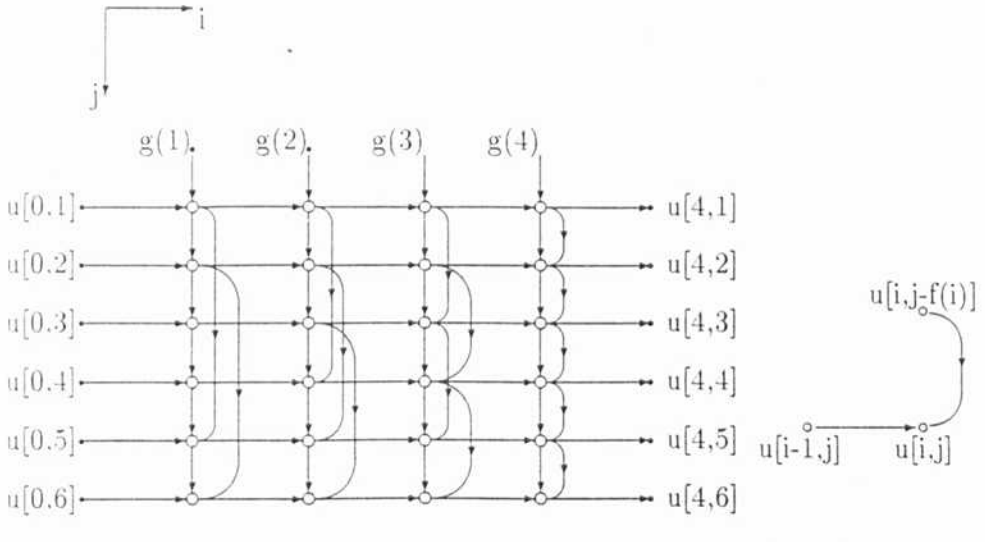
Следният пример илюстрира разглеждания подход.

Пример 1 *Нека* $l = 0$ *или* $l = 1$, $m = 4$, $c = 5$, $g(1) = 2$, $g(2) = 3$, $g(3) = 4$, $g(4) = 1$ $f(1) = 4$, $f(2) = 3$, $f(3) = 2$ $f(4) = 1$. *Съответните графи на зависимостите за* $l = 0$ *и* $l = 1$ *са дадени на фигура* 2.1*a* *и* 2.1*b*. *Зависимостта на стойностите* $u[i, j]$ *е показана на фигури* 2.1*c* *и* 2.1*d*.

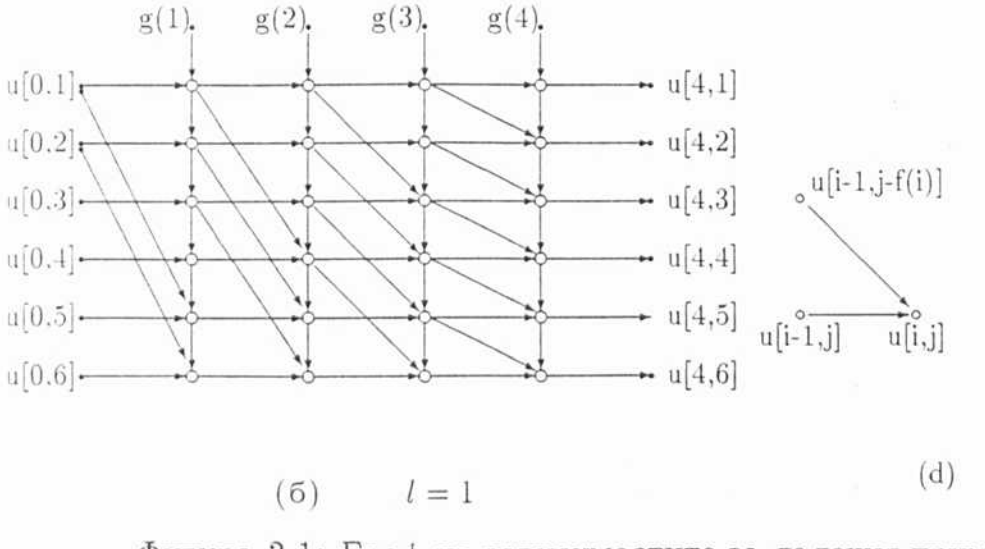
Графът на зависимостите има следните свойства, които го правят удобен за паралелна реализация.

- Във всеки връх се извършват операциите F или идентитет;
- Има само два вида зависимости на данните.

Изчисленията се извършват в нарастващ ред спрямо i докато $i = m$ и за дадено i - последователно в нарастващ ред спрямо j за $j = 0, \ldots, c$.

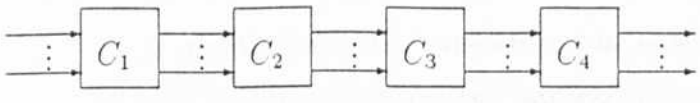


(а) $l = 0$



(б) $l = 1$

Фигура 2.1: Граф на зависимостите за дадения пример



Фигура 2.2: Линеен систолически масив

Целта в тази глава е да се намерят съответно времева функция и алокираща функция $(t(z),\ alloc(z))$, които да изобразяват графа на зависимостите върху линеен хардуерен граф (граф, който служи за представяне на линейния систолически маси).

2.2 Конструиране на линеен систолически масив

В тази секция разглеждаме реализация на задачи от типа на задачата за раницата (описани с уравнение (2.1)) върху линеен систолически масив (фигура 2.2). Линейният систолически масив, състоящ се от p изчислителни елемента, може да се представи чрез граф $G(V, E)$. Ще номерираме върховете на графа с числата от 1 до p. Между два върха $v, w \in V$ има ребро, само ако номерата им се различават с 1.

2.2.1 Техника на изобразяването

Първо ще бъде определена алокираща функция, а след това валидна времева функция.

Ще използваме метода на покритията разгледан в [18] за намирането на алокираща функция. Ще дефинираме графи G_i, $i = 1, \ldots, m - l$,

по следния начин. Ако $f(i+l) \leq \frac{c}{2}$, то графът G_i има $\left\lceil \frac{f(i+l)}{\alpha} \right\rceil$ нетерминални върха, номерирани с целите числа от 1 до $\left\lceil \frac{f(i+l)}{\alpha} \right\rceil$. Ребрата на графа отговарят на следното условие. От връх с номер r излиза ребро към връх с номер $r+1$, за $r = 1, \ldots, \left\lceil \frac{f(i+l)}{\alpha} \right\rceil - 1$. На връх (i,j) от графа на зависимостите съответства връх с номер k от граф G_i, където $k = \left\lceil \frac{j \bmod (f(i+l))+1}{\alpha} \right\rceil$, за $i = 1, \ldots, m-l$.

Ако $f(i+l) > \frac{c}{2}$, графът G_i, за $i = 1, \ldots, m$ има $\left\lceil \frac{c-f(i+l)+1}{\alpha} \right\rceil$ нетерминални върха, номерирани с целите числа от 1 до $\left\lceil \frac{c-f(i+l)+1}{\alpha} \right\rceil$. Ребрата на графа отговарят на следното условие. От връх с номер r излиза реброо към връх с номер $r+1$, за $r = 1, \ldots, \left\lceil \frac{c-f(i+l)+1}{\alpha} \right\rceil - 1$. При $j \leq c - f(i+l)$ или $c \geq j \geq f(i+l)+1$, на връх (i,j) от графа на зависимостите съответства връх с номер k от G_i , където $k = \left\lceil \frac{j \bmod (f(i+l))+1}{\alpha} \right\rceil$, $i = 1, \ldots m$. При $f(i+l) > j \geq c - f(i+l)$, на връх (i,j) от графа на зависимостите съответства връх с номер k от G_i, където $k = \left\lceil \frac{c-f(i+l)+1}{\alpha} \right\rceil$, $i = 1, \ldots, m$.

С w_i означаваме броя на върховете на графа G_i, за $i = 1, \ldots, m$.

$$w_i = \begin{cases} \left\lceil \frac{f(i+l)}{\alpha} \right\rceil & \text{за } f(i+l) \leq \frac{c}{2} \\ \\ \left\lceil \frac{c-f(i+l)+1}{\alpha} \right\rceil & \text{за } f(i+l) > \frac{c}{2} \end{cases}, \; i = 1, \ldots, m-l.$$

При $l = 1$ $w_m = 1$.

С $Q(G_i)$ означаваме множеството на графите G_i, за $i = 1, \ldots, m$

Нека разгледаме линеен систолически масив, описан чрез линейния ориентиран граф H с $\sum_{i=1}^{m} w_i$ нетерминални върха. Нетерминалните върхове на графа H са номерирани с цели положителни числа започващи от 1 и изобразяват изчислителните елементи (клетките) на систолическия масив, а ребрата показват връзките между тях. Реброто влизащо в първата клетка ще наричаме входно ребро, а реброто изли-

защо от последната клетка ще наричаме изходно ребро. Връх с номер r е свързан чрез ребро с връх с номер $r+1$ ако съществува.

Целта ни е да изобразим множеството от графи $Q(G_i)$ върху линейния граф H.

Нека $Q(G_i)$ е множество от графите G_i и $V' = \cup V(G_i)$ е множество от върховете на $Q(G_i)$,а $E' = \cup E(G_i)$ е множеството от ребрата на $Q(G_i)$.

Ще дефинираме изображение $P : V' \to V(H)$. Нека (i,k) означава връх с номер k от графа G_i, $i = 1, \ldots, m$.

$$P((i,k)) = \begin{cases} \sum_{s=1}^{i-1} w_s + k & i > 1 \\ \\ k & i = 1 \end{cases},$$

където w_i е броят на върховете на G_i, за $i = 1, \ldots, m$, а дясната част на равенството е номер на връх от H.

Дефиниция 4 *Двойката $\{Q, P\}$ се нарича покритие на H, ако следните условия са удовлетворени:*

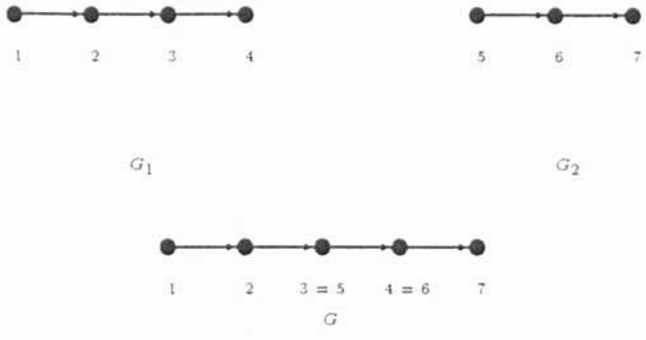
1. P е еднозначно обратимо изображение на V'_n върху $V_n(H)$.

2. Ако $(\bar{x}, \bar{y}) \in E'$, то $(P(\bar{x}), P(\bar{y})) \in E(H)$, според това условие P дефинира изображение на E' върху $E(H)$, което също ще означаваме с P, така $P(\bar{x}, \bar{y}) = (P(\bar{x}), P(\bar{y}))$

3. Ако $\bar{x} \in V'$ и реброто $e \in E(H)$ е инцидентно с $P(\bar{x})$, то съществува ребро $e' \in E'$ инцидентно с $\bar{x}$ и $P(e') = e$.

Теорема 1 *Двойката $\{Q, P\}$ удовлетворява условията на дефиниция 4.*

Доказателство. От дефиницията на изображението P следва, че условие 1 на дефиниция 4 е удовлетворено.



Фигура 2.3: Покритие на граф G с графите G_1 и G_2

Нека $(\bar{x}, \bar{y}) \in E'$, тогава $\bar{x} = (i,k)$ и $\bar{y} = (i,k+1)$, $1 \leq i \leq m$ $1 \leq k \leq w_i$ и

$$P(\bar{x}) = P((i,k)) = \begin{cases} k & i = 1 \\ \\ \sum_{s=1}^{i-1} w_s + k & i > 1 \end{cases},$$

$$P(\bar{y}) = P((i,k+1)) = \begin{cases} k+1 & i = 1 \\ \\ \sum_{s=1}^{i-1} w_s + k + 1 & i > 1 \end{cases}.$$

Следователно $(P(\bar{x}), P(\bar{y})) \in E(H)$, и условие 2 на дефиниция 4 е удовлетворено.

Нека $\bar{x} \in V'$ т.е. $\bar{x} = (i,k)$, $1 \leq i \leq m$, $1 \leq k \leq w_i$, тогава

$$P(\bar{x}) = P((i,k)) = \begin{cases} k & i = 1 \\ \\ \sum_{s=1}^{i-1} w_s + k & i > 1 \end{cases},$$

а реброто

$$e = \begin{cases} (k, k+1) & k = 1 \\ \\ \left(\sum_{s=1}^{i-1} w_s + k, \sum_{s=1}^{i-1} w_s + k + 1\right) & i > 1 \end{cases}$$

е инцидентно с връх $P(\bar{x})$. Нека ребро $e' = (\bar{x}, \bar{y}) \in E'$ и $\bar{y} = (i, k+1)$, следователно

$$P(\bar{y}) = P((i,k+1)) = \begin{cases} k+1 & i = 1 \\ \\ \sum_{s=1}^{i-1} w_i + k + 1 & i > 1 \end{cases},$$

или $P(e') = e$, т.е. условие 3 на дефиниция 4 е удовлетворено. Така двойката $\{Q, P\}$ е покритие за H. □

2.2.2 Алокираща функция и времева функция

Чрез изображението P определяме следната алокираща функция:

$$alloc(i,j) = P(i,j) = \begin{cases} k(i,j) & i = 1 \\ \\ \sum_{s=1}^{i-1} w_s + k(i,j) & i > 1 \end{cases}, \tag{2.2}$$

$$\text{където } k(i,j) = \begin{cases} \left\lceil \frac{c-f(i+l)+1}{\alpha} \right\rceil & \text{за} f(i+l) > \frac{c}{2},\ f(i+l) \geq j > c - f(i+l) \\ \left\lceil \frac{j \bmod (f(i+l))+1}{\alpha} \right\rceil & \text{в останалите случаи} \end{cases}$$

Ще разгледаме изчисленията, които се извършват от една клетка. Стойността $u[i,j]$ се изчислява когато се получи стойността $u[i-1,j]$. Стойността $u[i,j]$ се пази в локалната памет на клетката в регистър с номер $adr = (j \bmod(f(i)) + 1) \bmod(\alpha)$ за да бъде използвана за изчисляването на стойността $u[i+l, j+f(i+l)]$.

Разглеждаме следната времева функция:

$$t(i,j) = \begin{cases} j + k(i,j) & i = 1 \\ \sum_{s=1}^{i-1} w_s + k(i,j) + j & 2 \leq i \leq m \\ \sum_{s=1}^{i-1} w_s + j & i = m+1 \end{cases}, \tag{2.3}$$

$$\text{където } k(i,j) = \begin{cases} \left\lceil \frac{c-f(i+l)+1}{\alpha} \right\rceil & \text{за} f(i+l) > \frac{c}{2},\ f(i) \geq j > c - f(i) \\ \left\lceil \frac{j \bmod (f(i))+1}{\alpha} \right\rceil & \text{в останалите случаи} \end{cases}.$$

Теорема 2 *Функцията $t(i,j)$ дефинирана в (2.3) е валидна времева функция.*

Доказателство. Нека $x = (i,j)$ и $y = (i',j')$, $x \neq y$ са върхове от графа на зависимостите, за които $alloc(x) = alloc(y)$. Съгласно определението за алокираща функция:

$$\begin{cases} k(i,j) & i=1 \\ \sum_{s=1}^{i-1} w_s + k(i,j) & i>1 \end{cases} = \begin{cases} k(i',j') & i'=1 \\ \sum_{s=1}^{i'-1} w_s + k(i',j') & i'>1 \end{cases}.$$

Ще докажем, че $t(x) \neq t(y)$. Нека допуснем, че $i \neq i'$ и $i > i'$, тогава: $\sum_{s=1}^{i-1} w_s + k(i,j) = \sum_{s=1}^{i'-1} w_s + \sum_{s=i'}^{i-1} w_s + k(i,j) = \sum_{s=1}^{i'-1} w_s + k(i',j') + \sum_{s=i'}^{i-1} w_s + k(i,j) - k(i',j') = \sum_{s=1}^{i'-1} w_s + k(i',j')$.

Следователно

$\sum_{s=i'}^{i-1} w_s + k(i,j) - k(i',j') = 0$,

По дефиниция $w_{i'} = \begin{cases} \left\lceil \frac{f(i'+l)}{\alpha} \right\rceil & f(i'+l) \leq \frac{c}{2} \\ \left\lceil \frac{c-f(i'+l)+1}{\alpha} \right\rceil & f(i'+l) > \frac{c}{2} \end{cases}$,

тогава $k(i',j') \leq w_{i'}$ и следователно $\sum_{s=i'}^{i-1} w_s + k(i,j) - k(i',j') > 0$. Стигнахме до противоречие, породено от допускането, че $i \neq i'$. Следователно $i = i'$ щом $alloc(x) = alloc(y)$.

$$t(i,j) = \begin{cases} j + k(i,j) & i=1 \\ \sum_{s=1}^{i-1} w_s + k(i,j) + j & 2 \leq i \leq m \\ \sum_{s=1}^{i-1} w_s + j & i = m+1 \end{cases} = \begin{cases} alloc(x) + j & i \leq m \\ \sum_{s=1}^{i-1} w_s + j & i = m+1 \end{cases}.$$

$$t(i,j') = \begin{cases} alloc(x) + j' & i \leq m \\ \sum_{s=1}^{i-1} w_s + j' & i = m+1 \end{cases}.$$

По условие имаме, че $alloc(x) = alloc(y)$ и $x \neq y$, получихме $i = i'$, следователно $j \neq j'$, а от там и $t(x) \neq t(y)$.

Нека $(x, y) \in E(G)$, където G е графът на зависимостите за уравнение (2.1). От уравнение 2.1 следва, че $y = (i-1, j)$ или $y = (i-l, j-f(i))$.

Нека $y = (i-1, j)$.

$t(i-1, j) = \sum_{s=1}^{i-2} w_s + k(i-1, j) + j$,

$t(i, j) = \sum_{s=1}^{i-1} w_s + k(i, j) + j = t(i-1, j) + w_{i-1} - k(i-1, j) + k$,

но $w_{i-1} \geq k(i-1, j)$, следователно $t(i, j) > t(i-1, j)$.

Нека $y = (i-l, j-f(i))$.

$t(i-l, j-f(i) = \sum_{s=1}^{i-l-1} w_s + k(i-l, j-f(i)) + j - f(i)$.

Ако $l = 1$,

$t(i, j) = \sum_{s=1}^{i-1} w_s + k(i, j) + j = t(i-1, j-f(i)) + w_{i-1} - k(i-1, j-f(i)) + k(i, j) + f(i)$,

$w_{i-1} > k(i-1, j-f(i))$ следователно $t(i, j) > t(i-l, j-f(i))$.

Ако $l = 0$.

$t(i, j) = t(i, j-f(i)) + k(i, j) - k(i, j-f(i)) + f(i)$.

От определението за $k(i, j)$ имаме $k(i, j) - f(i) \leq f(i)$, следователно $t(i, j) > t(i, j-f(i))$.

С това теоремата е доказана. □

От тази теорема следва:

1. В даден момент време, в дадена клетка се изчислява най-много една стойност $u[i, j]$, $i = 1, \ldots, m$ $j = 1, \ldots, c$

2. Когато изчисляваме стойността $u[i, j]$, всички стойности от които тя зависи са вече изчислени.

2.3 Функциониране на отделната клетка

В тази секция разглеждаме функционирането на отделната клетка, в линейния систолически масив. Стойността $u[i, j]$ се изчислява когато се

получи стойността $u[i-1,j]$. Според алокиращата функция, стойността $u[i,j]$ е необходима след $alloc(i+1,j) - alloc(i,j) = w_i - k(i,j) - k(i+1,j)$ клетки, в клетката където ще се изчислява $u[i+1,j]$. Където

$$k(i,j) = \begin{cases} \left\lceil \frac{c-f(i+l)+1}{\alpha} \right\rceil & \text{за} f(i+l) > \frac{c}{2},\ f(i+l) \geq j > c - f(i+l) \\ \left\lceil \frac{j \bmod (f(i+l))+1}{\alpha} \right\rceil & \text{в останалите случаи} \end{cases}$$

Стойността $u[i,j]$ се запазва в регистър от локалната памет с номер $adr = (j \bmod (f(i)) + 1) \bmod (\alpha)$ на клетка от систолическият масив с номер равен на $alloc(i,j)$, за да бъде използвана за изчисляването на стойността $u[i+l, j+f(i+l)]$, за $i+l \leq m,\ j+f(i+l) \leq c$. Стойностите $u[0,j]$ влизат в първата клетка на систолическия масив в момент j, $j = 1, \ldots, c$. Стойностите $u[m,j]$ излизат от последната клетка на линейния систолически масив в момент:

$t(m+1,j) = \sum_{s=1}^{m} w_s + j$, където

$$w_i = \begin{cases} \left\lceil \frac{f(i+l)}{\alpha} \right\rceil & f(i+l) \leq \frac{c}{2} \\ \left\lceil \frac{c-f(i+l)+1}{\alpha} \right\rceil & f(i+l) > \frac{c}{2} \end{cases},$$

Така в момент

$$T = \sum_{s=1}^{m} w_s + c,$$

всички стойности $u[m,j]$ вече са получени. Получаваме, че алгоритъмът за решаване на уравнение 2.1 се реализира за време

$$T = \sum_{s=1}^{m} w_s + c,$$

върху $p = \sum_{s=1}^{m} w_s$ изчислителни елемента.

Реализацията на пример 1 върху линеен систолически масив с локална памет $\alpha = 1$ регистъра е показана на фигура 2.4, където ● означава извършване на изчисления, а ○ предаване на данните без извършване на изчисления.

На фигура 2.5 сме представили схематично отделната клетка, от която е съставен линейният систолически масив.

Стойностите $g(i)$ и $f(i)$ влизат в клетките (фигура 2.5) с номера от $\sum_{s=1}^{i-1} w_s + 1$ до $\sum_{s=1}^{i} w_s$, $i = 2, \ldots, m$, при $i = 1$ тези стойности влизат в клетки с номера от 1 до w_1, на един предварителен етап и се запазват съответно в регистри g и f, това може да стане и в първият момент едновременно с влизането на стойността $u[0,1]$ в първата клетка. Стойността $f(i+1)$ се зарежда в регистър f_1, за $i = 1, \ldots, m-1$, за $i = m$ в регистър f_1 се зарежда стойност 0.

От "ляво" на всяка клетка влизат следните променливи:

- u - стойността на $u[i-1, j]$ за текущите i и j;

- j - текуща стойност на променливата j;

- i - текуща стойност на променливата i;

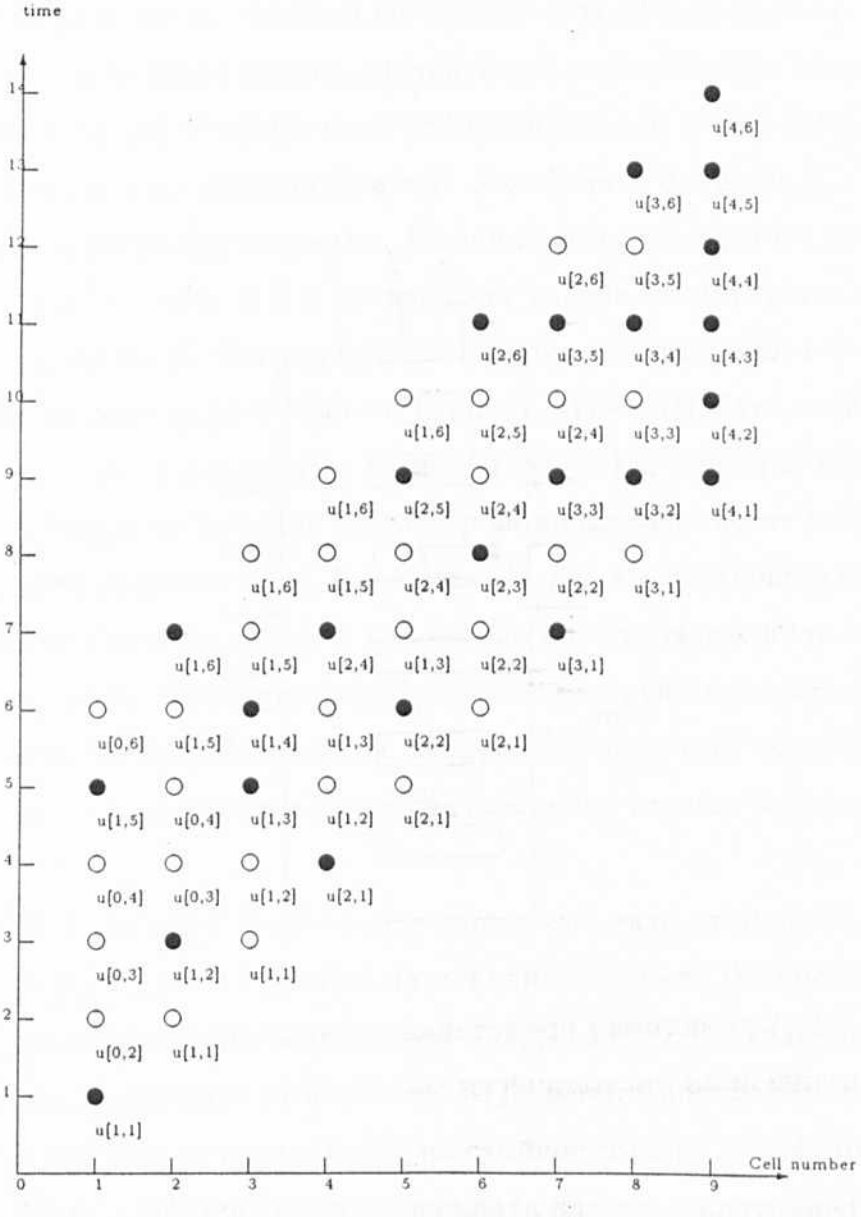
- Num - след колко клетки се намира клетката в която стойността $u[i,j]$ трябва да бъде изчислена;

- $u1$ - стойността на $u[i-l, j-f(i)]$.

В първата клетка на линейния систолически масив във всеки момент влиза по една стойност $u[0,j]$, $j = 1, \ldots, c$, започвайки от $u[0,1]$ в нарастващ ред спрямо j. Едновременно с всяка стойност $u[0,j]$ влиза стойността j, която не се променя при преминаването си през масива. В този момент влизат и стойностите:

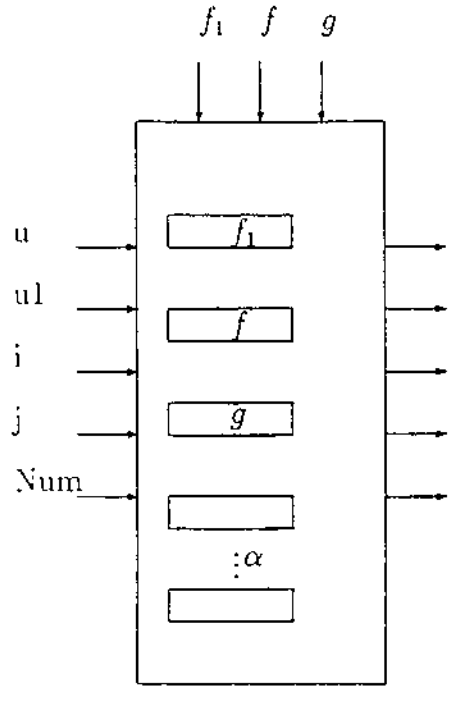
- $Num = alloc(1,j)$

- $i = 0$



Фигура 2.4: Реализация на пример 1 върху линеен систолически масив при $\alpha = 1$

11



Фигура 2.5: Схематично представяне на отделната клетката от която е съставен линейния систолически масив

- $u1 = 0$

Нека $l = 0$. В този случай променливата $u1$ не е необходима.

Ще разгледаме изчисляването на стойностите $u[m, j]$ за $j = 1, \ldots, c$.

В момент j в първата клетка на линейния систолически масив постъпва стойността $u[0, j]$ заедно със стойностите $i = 0$, $Num = alloc(1, j)$ и j. Стойността на Num се намалява с 1. Ако $Num > 0$, то $u[0, j]$, i, Num, j преминават в следващата клетка. Намаляването на Num с 1 и предаването на $u[0, j]$, i, Num, и j в следващата клетка се извършва докато Num стане равно на 0. Тогава i се увеличава с 1, в случая $i = 1$. Ако $j > f(i)$ изчисляваме $u[i, j] = F(u[i-1, j], u[i, j-f(i)]+g(i))$, като стойността $u[i, j - f(i)]$ се чете от регистър с номер $(j \bmod(f(i)) + 1) \bmod(\alpha)$ на локалната памет. След това в същия регистър на локалната памет се записва новополучената стойност $u[i, j]$ (в случая $u[1, j]$). На променливата Num присвояваме стойността $alloc(i + 1, j) - alloc(i, j)$ указваща колко клетки трябва да премине стойността $u[i, j]$, докато достигне клетката в която ще се използва за изчисляване на стойността $u[i + 1, j]$. Стойностите $u[i, j]$, i, Num, j преминават в следващата клетка, където операциите се повтарят.

Ако $j < f(i)$, то $u[i, j] = u[i - 1, j]$ и записваме тази стойност в регистър с номер $(j \bmod(f(i)) + 1) \bmod(\alpha)$ на локалната памет. На променливата Num присвояваме стойността $alloc(i + 1, j) - alloc(i, j)$ указваща колко клетки трябва да премине стойността $u[i, j]$, докато достигне клетката в която ще се използва за изчисляване на стойността $u[i+1, j]$. Стойностите $u[i, j]$, i, Num, j преминават в следващата клетка, където операциите се повтарят.

Така в момент $t(m, j)$ стойностите $u[m - 1, j]$, $i = m - 1$, $Num = 1$, j влизат в клетка с номер $alloc(m, j)$. Намаляваме променливата Num с 1 и тя приема стойност 0. Увеличаваме стойността на i с 1 и i приема

стойността m.

Ако $j > f(i)$ изчисляваме $u[m,j] = F(u[m-1,j], u[m,j-f(m)] + g(m))$, като стойността $u[m,j-f(i)]$ четем от регистър с номер $(j \mathrm{mod}(f(m)) + 1) \mathrm{mod}(\alpha)$ на локалната памет. Тази стойност е изчислена и записана в локалната памет в момент $t(m,j-f(m))$. След това в същия регистър на локалната памет записваме стойността $u[m,j]$. На променливата Num присвояваме стойността -1, тъй като $u[m,j]$ е една от търсените стойности. По този начин $u[m,j]$, $i = m$, j, Num преминават през оставащите клетки от систолическия масив, без извършване на изчисления.

Ако $j < f(m)$ то $u[m,j] = u[m-1,j]$ и записваме тази стойност в регистър с номер $(j \mathrm{mod}(f(m))+1) \mathrm{mod}(\alpha)$ на локалната памет. На променливата Num присвояваме стойността -1, защото $u[m,j]$ е една от търсените стойности. По този начин стойностите $u[m,j]$, $i = m$, j, Num преминават през оставащите $w_m - k(m,j)$ клетки от систолическия масив, без извършване на изчисления.

Следната програма, написана на псевдо-програмен език, показва как работи отделната клетка.

Начало:

$Num = Num - 1$

$if(Num = 0)$

$if(f < j)$

$u = F(u, u1 + g)$ / $u1$ е стойността, записана в регистър с номер $(j \mathrm{mod} f + 1) \mathrm{mod} \alpha)$ след това в същия регистър записваме новополучената стойност u/

$else$

u-не се променя, записваме неговата стойност в регистър с номер $(j \mathrm{mod} f + 1) \mathrm{mod} \alpha$

$endif$

$i = i + 1$

$Num = alloc(i, j) - alloc(i - 1, j)$

$endif$

$if(i + 1 > m)$

$Num = -1$

$endif$

Край.

Ще разгледаме изчисляването на стойността $u[m, j]$ за $j = 1, \ldots, c$.

В момент j в първата клетка на линейния систолически масив постъпва стойността $u[0, j]$, заедно със стойностите $i = 0$, $Num = alloc(1, j)$, j и $u1 = 0$. Стойността на Num се намалява с 1. Ако $Num > 0$, то $u[0, j]$, i, Num, j и $u1$ преминават в следващата клетка. Намаляването на Num с 1 и предаването на $u[0, j]$, i, Num, j и $u1$ в следващата клетка се извършва докато Num стане 0. Тогава i се увеличава с 1, в случая $i = 1$. Ако $j > f(2)$ на $u1$ присвояваме стойността $u[0, j - f(1)]$ намираща се в регистър с номер $((j - f(1))\mathrm{mod} f(1) + 1)\mathrm{mod}(\alpha)$ на локалната памет и в същият регистър записваме стойността $u[0, j]$. На променливата Num присвояваме стойността $alloc(2, j) - alloc(1, j)$ и стойностите $u[1, j] = u[0, j]$, $u1$, i, j и Num преминават в следващата клетка.

Ако $j < f(2)$ записваме стойността на $u[1, j] = u[0, j]$ в регистър с номер $((j - f(1))\mathrm{mod} f(1) + 1)\mathrm{mod}(\alpha)$ на локалната памет. На променливата Num присвояваме стойността $alloc(2, j) - alloc(1, j)$ и стойностите $u[1, j] = u[0, j]$, $u1$, i, j и Num преминават в следващата клетка.

В тази клетка (в случая 2) стойността на Num се намалява с 1. Ако $Num > 0$, то $u[i, j]$, $u1$, i, j и Num преминават в следващата клетка. Намаляването на Num с 1 и предаването на следващата клетка се извършва докато Num стане равно на 0. Тогава i се увеличава с 1. Ако $j > f(i)$ изчисляваме стойността $u[i, j] = F(u[i-1, j], u[i-1, j-f(i)] + g(i))$,

като $u[i-1, j-f(i)]$ е стойността на променливата $u1$. На $u1$ присвояваме стойността от регистър с номер $((j)\mathrm{mod}f(i)+1)\mathrm{mod}(\alpha)$ и в същия регистър записваме стойността $u[i,j]$. На променливата Num присвояваме стойността $alloc(i+1,j) - aiilc(i,j)$ и стойностите $u[i,j]$, $u1$, i, j и Num преминават в следващата клетка, където операциите се повтарят.

Ако $j < f(i)$ записваме стойността $u[i,j] = u[i-1,j]$ в регистър с номер $((j-f(i))\mathrm{mod}f(i)+1)\mathrm{mod}(\alpha)$ на локалната памет. На променливата Num присвояваме стойността $alloc(i+1,j) - alloc(i,j)$ и стойностите $u[i,j]$, $u1$, i, j, и Num преминават в следващата клетка, където операциите се повтарят.

Така в момент $t(m,j)$ стойностите $u[m-1,j]$, $u1 = u[m-1, j-f(m)]$, $i = m-1$, j и $Num = 1$ влизат в клетка с номер $alloc(m,j)$, това е последната клетка. Намаляваме Num с 1 и Num става равно на 0. Увеличаваме стойността на i с 1 и i приема стойността m. Ако $j > f(m)$ изчисляваме стойността $u[m,j] = F(u[m-1,j], u[m-1, j-f(m)] + g(m))$, това е една от търсените стойности. Ако $j < f(m)$ то $u[m,j] = u[m-1,j]$.

Следната програма, написана на псевдо-програмен език, показва как работи отделната клетка.

Начало:

$Num = Num - 1$

$if(Num = 0)$

$if(f < j)$

$u = F(u, u1 + g)$

На $u1$ присвояваме стойността от регистър $(j\mathrm{mod}f(i)+1)\mathrm{mod}\alpha$ и в същия регистър записваме стойността на u

$else$

u-не се променя, записваме го в регистър с номер $(j\mathrm{mod}f(i)+1)\mathrm{mod}\alpha$

endif

$i = i + 1$

$Num = alloc(i+1, j) - alloc(i, j)$

endif

Край.

Теорема 3 *На изхода на така описания линеен систолически масив се получават необходимите стойности* $u[m, j]$, *за* $j = 1, \ldots, c$.

Доказателство. За алокиращата функция имаме:

$$alloc(i+1,j)-alloc(i,j) = \begin{cases} k(i+1,j) - k(i,j) & i = 1 \\ \sum_{s=1}^{i} w_s + k(i+1,j) - \sum_{s=1}^{i-1} w_s - k(i,j) & i > 1 \end{cases} =$$

$$= \begin{cases} k(i+1,j) - k(i,j) & i = 1 \\ w_i + k(i+1,j) - k(i,j) & i > 1 \end{cases}, \; i = 1, \ldots, c, \; j = 1, \ldots, m.$$

За времевата функция имаме:

$$t(i+1,j) - t(i,j) = \begin{cases} k(i+1,j) - k(i,j) & i = 1 \\ \sum_{s=1}^{i} w_s + k(i+1,j) - \sum_{s=1}^{i-1} w_s - k(i,j) & i > 1 \end{cases} =$$

$$= \begin{cases} k(i+1,j) - k(i,j) & i = 1 \\ w_i + k(i+1,j) - k(i,j) & i > 1 \end{cases}, \; i = 1, \ldots, c, \; j = 1, \ldots, m.$$

Следователно $t(i+1, j) - t(i, j) = alloc(i+1, j) - alloc(i, j)$.

Това означава, че стойността $u[i, j]$ постъпва в клетка с номер $alloc(i+1, j)$ в момента в който ще изчисляваме стойността $u[i+1, j]$. От това,

че времевата функция е валидна времева функция следва, че на входа на всяка клетка в даден момент време влиза точно една стойност $u[i,j]$, т.е. няма конфликт на данните.

При $l = 0$ стойността $u[i+1, j-f(i)]$ е вече изчислена, следва от валидността на времевата функция, и се намира в същата клетка в регистър с номер равен на $(j \mathrm{mod} f(i) + 1) \mathrm{mod} \alpha$. Така в тази клетка изчисляваме $u[i+1,j] = F(u[i,j], u[i+1, j-f(i+1)] + g(i))$. Следователно на изхода получаваме необходимата стойност $u[m,j]$.

При $l = 1$ от алокиращата и от времевата функции следва, че в момент $t(i,j)$ в клетка с номер равен на $alloc(i,j)$ се изчислява стойността $u = u[i,j]$, а стойността $u1 = u[i, j-f(i+1)]$ се намира в регистър с номер равен на $(j \mathrm{mod}(f(i+1))+1) \mathrm{mod}(\alpha)$ на същата клетка. И двете стойности се предават на следващата клетка. В момент $t(i+1,j)$ тези стойности достигат до клетка с номер равен на $alloc(i+1,j)$ в която се изчислява стойността $u[i+1,j] = F(u[i,j], u[i, j-f(i+1)] + g(i+1))$. От времевата функция следва, че няма конфликт на данни. Следователно на изхода получаваме необходимите стойности $u[m,j]$. □

2.4 Линеен систолически масив с фиксиран брой изчислителни елементи

До сега предполагахме, че разполагаме с достатъчен брой клетки за конструиране на линеен систолически масив.

В тази секция разглеждаме линеен систолически масив съставен от p изчислителни елемента (клетки), на които последната клетка е свързана с първата. Без ограничение на общността допускаме, че $\sum_{i=1}^{m} w_i > p$, ако $p > \sum_{i=1}^{m} w_i$, клетките след $\sum_{i=1}^{m} w_i$ не се използват за изчисляване на

стойностите $u[i,j]$. В случая, линейният масив е пръстен съставен от p клетки от същия тип както в предходната секция, с опашка с размер a, където

$$a = \begin{cases} c \dot{-} p & c > p \\ 1 & c \leq p \end{cases},$$

която представлява памет от тип $FIFO$ (първият влязъл е първия излязъл) и се използва за прехвърляне на данните от последната към първата клетка. Стойностите $u[i,j]$ се разпределят в $r = \left\lceil \frac{\sum_{i=1}^{m} w_i}{p} \right\rceil$ множества B_d, които се обработват последователно в нарастващ ред спрямо d. $d = 1, \ldots, r$, $u[i,j] \in B_d$ точно тогава когато $\left\lceil \frac{alloc(i,j)}{p} \right\rceil = d$. Всяко множество B_d, $d = 1, \ldots, r$, се реализира върху p изчислителни елементи, както беше разгледано в предходната секция, т.е. всяко множество се реализира върху пръстена с времева сложност $t_d = c + p$. Множествата B_d са $r = \left\lceil \frac{\sum_{i=1}^{m} w_i}{p} \right\rceil$ на брой и се реализират последователно, т.е. когато последната от стойностите на B_d е във втората клетка на пръстена, в първата клетка се намира първата стойност от B_{d+1} за $d = 1, \ldots, r-1$, следователно времевата сложност на цялата задача върху пръстен с p изчислителни елемента е:

$$T = c \left\lceil \frac{\sum_{i=1}^{m} w_i}{p} \right\rceil + p \leq c \frac{\sum_{i=1}^{m} \left\lceil \frac{f(i)}{\alpha} \right\rceil}{p} + p.$$

2.5 Заключение

В тази глава разгледахме реализацията на нехомогенни рекурентни уравнения за задачи от типа на задачата за раницата върху линеен сис-

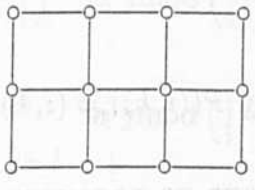
толически масив. Първо описахме реализацията на задачата за раницата върху линеен систолически масив, при който разполагаме с толкова клетки, колкото са ни необходими. В случая когато локалната памет е един регистър и всички стойности $f(i)$ са равни, за $i = 1, \ldots, m$, нашият резултат се изравнява с този на Andonov и Benaini [3]. В останалите случаи получаваме по-добър резултат, както за броя на изчислителните елементи, така и за времевата сложност на реализацията. Доказахме, че на изхода на линейния систолически масив се получават необходимите стойности. Описахме реализация на задачата, когато линейният систолически масив има фиксиран брой изчислителни елементи. Когато всички стойности на $f(i)$, за $i = 1, \ldots, m$ са равни и локалната памет $\alpha = 1$, времевата сложност на задачата се изравнява с тази на Andonov и Quinton [5] при равен брой изчислителни елементи. В останалите случаи получаваме по-добър резултат.

Глава 3

Двумерна реализация на задачи от типа на задачата за раницата

В тази глава разглеждаме реализацията на задачи от типа на задачата за раницата, описани с уравнение (2.1), върху двумерен правоъгълен систолически масив. Двумерният правоъгълен систолически масив, състоящ се от $p = n \times q$ изчислителни елемента, се представя, чрез граф $G(V, E)$. На всеки връх съпоставяме наредена двойка числа (i, j), $i = 1, \ldots, n$, $j = 1, \ldots, q$. Между два върха $v = (i, j), w = (i', j') \in V$ има ребра, само ако $|i - i'| + |j - j'| = 1$

Подобно на предходната глава ще използваме метода на покрития-



Фигура 3.1: Двумерен правоъгълен систолически масив

та за намиране на алокираща функция. Ще дефинираме графи G_i, за $i = 1, \ldots, m$, по следния начин. Графът G_i има $\left\lceil \frac{f_{max}}{\alpha} \right\rceil$, $f_{max} = \max_i f(i)$, нетерминални върха, номерирани с целите положителни числа от 1 до $\left\lceil \frac{f_{max}}{\alpha} \right\rceil$. От връх с номер r излиза ребро към връх с номер $r+1$, за $r = 1, \ldots, \left\lceil \frac{f_{max}}{\alpha} \right\rceil - 1$. На връх (i,j) от графа на зависимостите съответства връх с номер k от граф G_i така,че

$$k = \begin{cases} j \bmod \left(\left\lceil \frac{f(i)}{\alpha} \right\rceil \right) & \text{за } j \bmod \left(\left\lceil \frac{f(i)}{\alpha} \right\rceil \right) \neq 0 \\ \left\lceil \frac{f(i)}{\alpha} \right\rceil & \text{за } j \bmod \left(\left\lceil \frac{f(i)}{\alpha} \right\rceil \right) = 0 \end{cases}, \text{ за } i = 1, \ldots, m$$

С $Q(G_i)$ означаваме множеството на графите G_i, за $i = 1, \ldots, m$, $V' = \cup_{i=1}^{m} V(G_i)$ е множеството от върховете на $Q(G_i)$, а $E' = \cup_{i=1}^{m} E(G_i)$ - е множеството от ребрата на G_i.

Нека разгледаме двумерен правоъгълен систолически масив изобразен чрез граф H с $m \times \left\lceil \frac{f(i)}{\alpha} \right\rceil$ върха. На всеки нетерминален връх съпоставяме наредена двойка числа (i,j), $i = 1, \ldots, m$ $j = 1, \ldots \left\lceil \frac{f_{max}}{\alpha} \right\rceil$. Тези върхове изобразяват изчислителните елементи на систолическия масив, а ребрата показват връзките между тях. Върховете $v = (i,j), w = (i',j') \in H$ са свързани с ребро, само ако $|i - i'| + |j - j'| = 1$.

Целта ни е да изобразим множеството от графи $Q(G_i)$ върху H.

Ще дефинираме изображението $P : V' \to V(H)$ по следния начин:

$$P((i,k)) = (i,k),$$

където (i,k) от лявата страна на равенството е връх с номер k от граф G_i, а дясната страна на равенството е връх (i,k) от граф H.

Теорема 4 *Двойката $\{Q, P\}$ е покритие за H*

Доказателство. От дефиницията на изображението P следва, че условие 1 на дефиниция 4 е удовлетворено.

Нека $(\bar{x},\bar{y}) \in E' = \cup_{i=1}^{m} E(G_i)$, тогава $\bar{x} = (i,k)$ и $\bar{y} = (i,k+1)$, за $1 \leq i \leq m$, $1 \leq k < \lceil \frac{f_{max}}{\alpha} \rceil$ и $P(\bar{x}) = P((i,k)) = (i,k)$, $P(\bar{y}) = P((i,k+1)) = (i,k+1)$. Имаме $|i-i| + |k+1-k| = 1$, следователно, от определението за правоъгълен систолически масив, двата върха са свързани с ребро, т.е. $P(\bar{x},\bar{y}) \in E(H)$ и условие 2 на дефиниция 4 е удовлетворено.

Нека $\bar{x} \in V'$, т.е. $\bar{x} = (i,k)$, $1 \leq i \leq m$, $1 \leq k \leq \lceil \frac{f_{max}}{\alpha} \rceil$, тогава $P(\bar{x}) = P((i,k)) = (i,k)$, а реброто $e = ((i,k),(i,k+1))$ е инцидентно с връх $P(\bar{x})$. Нека реброто $e' = (\bar{x},\bar{y}) \in E'$ и $\bar{y} = (i,k+1)$, следователно $P(\bar{y}) = P((i,k)) = (i,k+1)$ или $(P(\bar{x}),P(\bar{y})) = e$, т.е. условие 3 на дефиниция 4 е удовлетворено. Така двойката $\{Q,P\}$ е покритие за H.

□

3.1 Алокираща функция и времева функция

Чрез изображението P определяме следната алокираща функция:

$$alloc(i,j) = P(i,k) = (i,k(i,j)),$$

$$\text{където } k(i,j) = \begin{cases} j \bmod \left(\left\lceil \frac{f(i+l)}{\alpha} \right\rceil\right) & \text{за } j \bmod \left(\left\lceil \frac{f(i+l)}{\alpha} \right\rceil\right) \neq 0 \\ \left\lceil \frac{f(i+l)}{\alpha} \right\rceil & \text{за } j \bmod \left(\left\lceil \frac{f(i+l)}{\alpha} \right\rceil\right) = 0 \end{cases},$$

$$\text{за } i = 1,\ldots,m-l \quad j = 1,\ldots,c, \text{ за } l = 1$$

$$k(m,j) = \begin{cases} j \bmod \left(\left\lceil \frac{f(m)}{\alpha} \right\rceil\right) & \text{за } j \bmod \left(\left\lceil \frac{f(m)}{\alpha} \right\rceil\right) \neq 0 \\ \left\lceil \frac{f(m)}{\alpha} \right\rceil & \text{за } j \bmod \left(\left\lceil \frac{f(m)}{\alpha} \right\rceil\right) = 0 \end{cases}, \text{ за } j = 1,\ldots,c.$$

Ще разгледаме изчисленията, които се извършват от една клетка. Стойността $u[i,j]$ се изчислява когато се получи стойността $u[i-1,j]$. Стойността $u[i,j]$ се пази в локалната памет на клетката в регистър с номер равен на $(j \bmod(f(i))+1) \bmod(\alpha)$ за да бъде използвана за изчисляването на стойността $u[i+l, j+f(i+l)]$. Ще допуснем, че $f(i) \leq f(i+1)$, за цитираните в началото на тази глава задачи, реда на стойностите $f(i)$ не е от значение. С f_{max} ще означаваме $\max_i f(i)$, а с f_{min} ще означаваме $\min_i f(i)$.

Използваме следната времева функция:

$$t(i,j) = (i-1)\left\lceil \frac{f_{max}}{\alpha} \right\rceil + \left\lceil \frac{j}{\left\lceil \frac{f_{min}}{\alpha} \right\rceil} \right\rceil . \tag{3.1}$$

Теорема 5 *Функцията $t(i,j)$ разгледана в (3.1) е валидна времева функция.*

Доказателство. Нека $x = (i,j)$ и $y = (i',j')$, $x \neq y$ са върхове на графа на зависимостите за които $alloc(x) = alloc(y)$. Следователно:

$alloc(i,j) = (i, k(i,j)) = (i', k(i',j')) = alloc(i',j')$,

$$\text{където } k(i,j) = \begin{cases} j \bmod \left(\left\lceil \frac{f(i+l)}{\alpha} \right\rceil\right) & \text{за } j \bmod \left(\left\lceil \frac{f(i+l)}{\alpha} \right\rceil\right) \neq 0 \\ \left\lceil \frac{f(i+l)}{\alpha} \right\rceil & \text{за } j \bmod \left(\left\lceil \frac{f(i+l)}{\alpha} \right\rceil\right) = 0 \end{cases},$$

$$\text{за } i = 1, \ldots, m-l \quad j = 1, \ldots, c,$$

$$\text{за } l = 1 \; k(m,j) = \begin{cases} j \bmod \left(\left\lceil \frac{f(m)}{\alpha} \right\rceil\right) & \text{за } j \bmod \left(\left\lceil \frac{f(m)}{\alpha} \right\rceil\right) \neq 0 \\ \left\lceil \frac{f(m)}{\alpha} \right\rceil & \text{за } j \bmod \left(\left\lceil \frac{f(m)}{\alpha} \right\rceil\right) = 0 \end{cases}, \; j = 1, \ldots, c.$$

Следователно $i = i'$ и $j \bmod \left(\left\lceil \frac{f(i+l)}{\alpha} \right\rceil\right) = j' \bmod \left(\left\lceil \frac{f(i+l)}{\alpha} \right\rceil\right)$ следователно $j' = j + sf(i+l)$, където s е цяло число различно от 0. Така:

$$t(i,j) = (i-1)\left\lceil \frac{f_{max}}{\alpha} \right\rceil + \left\lceil \frac{j}{\left\lceil \frac{f_{min}}{\alpha} \right\rceil} \right\rceil,$$

$$t(i, j+sf(i)) = (i-1)\left\lceil \frac{f_{max}}{\alpha} \right\rceil + \left\lceil \frac{j+sf(i)}{\left\lceil \frac{f_{min}}{\alpha} \right\rceil} \right\rceil$$

и $t(i,j) \neq t(i,j')$.

Нека $(y,x) \in E(G)$, където G е графът на зависимостите за уравнение (2.1). От уравнение (2.1) следва,че $y = (i-1,j)$ или $y = (i-l, j-f(i))$.

Нека $y = (i-1,j)$

$$t(i-1,j) = (i-2)\left\lceil \frac{f_{max}}{\alpha} \right\rceil + \left\lceil \frac{j}{\left\lceil \frac{f_{min}}{\alpha} \right\rceil} \right\rceil,$$

$$t(i,j) = (i-1)\left\lceil \frac{f_{max}}{\alpha} \right\rceil + \left\lceil \frac{j}{\left\lceil \frac{f_{min}}{\alpha} \right\rceil} \right\rceil = t(i-1,j) + \left\lceil \frac{f_{max}}{\alpha} \right\rceil > t(i-1,j).$$

Нека $y = (i-l, j-f(i))$

$$t(i-l, j-f(i)) = (i-l-1)\left\lceil \frac{f_{max}}{\alpha} \right\rceil + \left\lceil \frac{j-f(i)}{\left\lceil \frac{f_{min}}{\alpha} \right\rceil} \right\rceil,$$

$$t(i,j) = (i-1)\left\lceil \frac{f_{max}}{\alpha} \right\rceil + \left\lceil \frac{j}{\left\lceil \frac{f_{min}}{\alpha} \right\rceil} \right\rceil =$$

$$t(i-l, j-f(i)) + l\left\lceil \frac{f_{max}}{\alpha} \right\rceil + \left\lceil \frac{j}{\left\lceil \frac{f_{min}}{\alpha} \right\rceil} \right\rceil - \left\lceil \frac{j-f(i)}{\left\lceil \frac{f_{min}}{\alpha} \right\rceil} \right\rceil > t(i-l, j-f(i)).$$

С това теоремата е доказана. □

От тази теорема следва:

1. В даден момент време в дадена клетка се изчислява точно една стойност $u[i,j]$, за $i = 1,\ldots,c$, $j = 1,\ldots,m$;

2. Когато изчисляваме стойността $u[i,j]$, всички стойности от които

тя зависи са вече изчислени.

3.2 Функциониране на отделната клетка

В тази секция разглеждаме функционирането на отделната клетка на двумерния правоъгълен систолически масив. Стойността $u[i,j]$ се изчислява, когато се получи стойността $u[i-1,j]$. Според алокиращата функция, клетката където ще се изчислява стойността $u[i,j]$ се намира в колона i на ред $k(i,j)$, където

$$k(i,j) = \begin{cases} j \bmod \left(\left\lceil \frac{f(i+l)}{\alpha} \right\rceil\right) & \text{за } j \bmod \left(\left\lceil \frac{f(i+l)}{\alpha} \right\rceil\right) \neq 0 \\ \left\lceil \frac{f(i+l)}{\alpha} \right\rceil & \text{за } j \bmod \left(\left\lceil \frac{f(i+l)}{\alpha} \right\rceil\right) = 0 \end{cases},$$

за $i = 1, \ldots, m-l \quad j = 1, \ldots, c$

$$\text{при } l = 1 \; k(m,j) = \begin{cases} j \bmod \left(\left\lceil \frac{f(m)}{\alpha} \right\rceil\right) & \text{за } j \bmod \left(\left\lceil \frac{f(m)}{\alpha} \right\rceil\right) \neq 0 \\ \left\lceil \frac{f(m)}{\alpha} \right\rceil & \text{за } j \bmod \left(\left\lceil \frac{f(m)}{\alpha} \right\rceil\right) = 0 \end{cases},$$

за $j = 1, \ldots, c$.

От това, че броят на клетките в един стълб на двумерния правоъгълен систолически масив е $\lceil \frac{f_{max}}{\alpha} \rceil$ следва, че са необходими не повече от $\lceil \frac{f_{max}}{\alpha} \rceil$ премествания на стойността $u[i-1,j]$ от клетката, където тя е изчислена, за достигане на клетката, където ще се изчислява стойността $u[i,j]$. Стойността $u[i,j]$ се запазва в регистър от локалната памет с номер $(j \bmod(f(i)) + 1) \bmod(\alpha)$ за да бъде използвана за изчисляването на стойността $u[i+l, j+f(i+l)]$, за $i+l \leq m, \quad j+f(i+l) \leq c$. Стойностите $u[0,j]$ влизат в първата клетка на систолическия масив в момент $\left\lceil \frac{j}{\left\lceil \frac{f_{min}}{\alpha} \right\rceil} \right\rceil$. Стойностите $u[m,j], \quad j = 1, \ldots, c$ излизат от клетките на пос-

ледния стълб на двумерния правоъгълен систолически масив в момент

$$t(m+1,j) = m\left\lceil \frac{f_{max}}{\alpha} \right\rceil + \left\lceil \frac{j}{\left\lceil \frac{f_{min}}{\alpha} \right\rceil} \right\rceil .$$

Така в момент

$$T = m\left\lceil \frac{f_{max}}{\alpha} \right\rceil + \left\lceil \frac{c}{\left\lceil \frac{f_{min}}{\alpha} \right\rceil} \right\rceil ,$$

всички стойности $u[m,j], \quad j = 1,\ldots,c$ вече са получени. Така получаваме, че алгоритъмът за решаване на уравнение 2.1 се реализира за време

$$T = m\left\lceil \frac{f_{max}}{\alpha} \right\rceil + \left\lceil \frac{c}{\left\lceil \frac{f_{min}}{\alpha} \right\rceil} \right\rceil ,$$

върху $p = m \times \left\lceil \frac{f_{max}}{\alpha} \right\rceil$ изчислителни елемента.

Стойностите $g(i), f(i), f(i+1)$ се зареждат в клетки с номера от $(i,1)$ до $(i, \left\lceil \frac{f_{max}}{\alpha} \right\rceil)$, $i = 1,\ldots,m$, и се запазват съответно в регистри g, f, f_1,, при $i = m$ то $f_1 = 0$, вида на отделната клетка е даден схематично на фигура 3.2

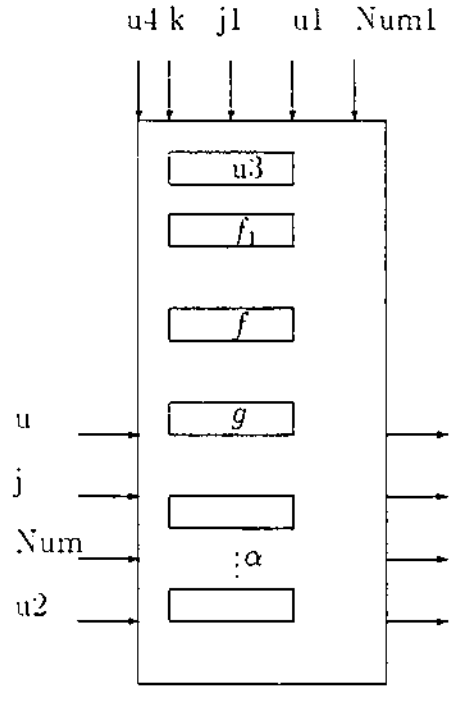
В двумерния правоъгълен систолически масив, който ще използваме, клетка $(i, \left\lceil \frac{f_{max}}{\alpha} \right\rceil)$ е свързана с клетка $(i,1)$ за $i = 1,\ldots,m$. Във всяка клетка влизат следните променливи:

- u - стойността $u[i-1,j]$,за текущите i и j, идваща от предходния стълб;

- $u1$ - стойността $u[i-1,j]$, за текущите i и j, придвижваща се по стълба до намиране на клетката в която се изчислява $u[i,j]$;

- $u2$ - стойността $u[i-l, j-f(i)]$, за текущите i и j, идваща от предходния стълб, тази променлива е необходима само при $l = 1$;

(ii)



Фигура 3.2: Схематично представяне на отделната клетката от която е съставен двумерният правоъгълен систолически масив

- $u4$ - стойността $u[i-l, j-f(i)]$, за текущите i и j, придвижващи се по стълба, до намиране на клетката в която се изчислява $u[i,j]$. Тази променлива е необходима само при $l = 1$;

- j - текуща стойност на променливата j, предавана между стълбовете;

- $j1$ - текуща стойност на променливата j, предавана по стълба;

- Num - след колко клетки се намира клетката в която стойността $u[i,j]$ трябва да бъде изчислена;

- $Num1$ - колко клетки трябва да се преминат по стълба;

- k - брояч, необходим за синхронизиране на предаването на данните между два съседни стълба.

В първият стълб на двумерния правоъгълен систолически масив, през $\lceil \frac{f_{max}}{\alpha} \rceil$ момента влизат $\lceil \frac{f_{min}}{\alpha} \rceil$ стойности, като от алокиращата функция следва, че в k-тата клетка на първия стълб влизат стойностите $u[0,j]$ за които $k = j \bmod \left(\left\lceil \frac{f(i)}{\alpha} \right\rceil \right) + 1$, в нарастващ ред спрямо j.

Едновременно с всяка стойност $u[0,j]$, влиза стойността j. В този момент влизат и стойностите:

- $Num = 1$

- $Num1 = 0$

- $j1 = 0$

- $u1 = 0$

- $k = 0$

Нека $l = 0$. Ще разгледаме изчисляването на стойността $u[m,j]$, за $j = 1, \ldots, c$.

В момент $\left\lceil \frac{j}{\left\lceil \frac{f_{min}}{\alpha} \right\rceil} \right\rceil$ в клетка $alloc(1,j) = (1, k(1,j))$,

$$k(1,j) = \begin{cases} j \bmod \left(\left\lceil \frac{f(1)}{\alpha} \right\rceil \right) & \text{за } \bmod \left(\left\lceil \frac{f(1)}{\alpha} \right\rceil \right) \leq 0 \\ \left\lceil \frac{f(1)}{\alpha} \right\rceil & \text{за } \bmod \left(\left\lceil \frac{f(1)}{\alpha} \right\rceil \right) = 0 \end{cases},$$

постъпва стойността $u[0,j]$ заедно със стойностите $Num = 1$, $Num1 = 0$, j, $j1 = 0$, $u1 = 0$, $k = 0$. Ако Num е различно от 0, намаляваме Num с 1, присвояваме на $u1$ стойността на $u[i,j]$ (в случая $u[0,j]$). Присвояваме на $Num1$ стойността на Num, а на k присвояваме стойността $\lceil \frac{f_{max}}{\alpha} \rceil - 1$. На $j1$ присвояваме стойността на j. Проверяваме дали $Num1$ е равно на 0. Ако $Num1$ е равно на 0 и $f(i) \geq j1 = j$, тогава $u[i+1,j] = u[i,j]$, (в случая $u[1,j]$) престоява k момента в регистър $u3$, k е брояч, необходим за синхронизацията на данните преминаващи през масива.

Ако $f(i) < j1 = j$, то изчисляваме стойността $u[i+1,j] = F(u[i,j], u[i, j - f(i)] + g(i))$. Стойността $u[i, j-f(i)]$ четем от регистър с номер $(j \bmod f(i) + 1) \bmod \alpha$ на локалната памет. В същият регистър записваме новополучената стойност $u[i+1,j]$. Намаляваме стойността на k с единица. Ако стойността на k е различна от 0, стойността на $u[i+1,j]$ престоява k момента в регистър с номер $u3$ на локалната памет. Ако $j \bmod \left\lceil \frac{f(i)}{\alpha} \right\rceil - j \bmod \left\lceil \frac{f(i+1)}{\alpha} \right\rceil \geq 0$, на променливата Num присвояваме стойността $j \bmod \left\lceil \frac{f(i)}{\alpha} \right\rceil - j \bmod \left\lceil \frac{f(i+1)}{\alpha} \right\rceil + 1$. В противен случай, на променливата Num присвояваме стойността $\lceil \frac{f_{max}}{\alpha} \rceil + j \bmod \left\lceil \frac{f(i)}{\alpha} \right\rceil - j \bmod \left\lceil \frac{f(i+1)}{\alpha} \right\rceil$. Стойността на променливата Num показва през колко клетки трябва да премине стойността $u[i,j]$ за да достигне клетката в който се изчислява стойността $u[i+1,j]$.

Ако $Num1$ е различно от 0, намаляваме стойността на $Num1$ с 1, намаляваме стойността на k с 1 и предаваме данните на съседните клетки, където операциите се повтарят.

Ако Num е равно на 0, предаваме данните на съседните клетки.

Така в момент $t(m,j)$, стойностите $u[m-1,j]$, $Num = 0$, $Num = 1$, $k = 1$, j, $j1 = j$, $u1 = u[m-1,j]$ влизат в клетка с номер $alloc(m,j)$ и изчисляваме стойността $u[m,j]$. Това е една от изходните стойности.

Следната програма, написана на псевдо-програмен език, показва как работи отделната клетка.

Начало:

$if(Num \neq 0)$

$Num = Num - 1$

$u1 = u$

$Num1 = Num$

$Num = 0$

$k = \lceil \frac{f_{max}}{\alpha} \rceil - 1$

$j1 = j$

$j = 0$

$endif$

$if(Num1 = 0)$

$if(f \geq j1)$

$u = u1$

$k = k - 1$

$if(k \neq 0)$

/стойността на променливата u престоява k момента в регистър $u3$/

$endif$

$endif$

$if(f < j1)$

$u = F(u1, u2 + g)$ /$u2$ е стойността записана в регистър с номер равен на $(j \bmod f(i)+1) \bmod \alpha$, в същия регистър се записва новополучената стойност u/

$u1 = 0$

$k = k - 1$

$if(k \neq 0)$

/стойността на променливата u престоява k момента в регистър $u3$/

$endif$

$endif$

$if\ ((j1\mathrm{mod}(\lceil \frac{f}{\alpha} \rceil) - j1\mathrm{mod}\ (\lceil \frac{f_1}{\alpha} \rceil) \geq 0)$

$Num = j1\mathrm{mod}(\lceil \frac{f}{\alpha} \rceil) - j1\mathrm{mod}(\lceil \frac{f_1}{\alpha} \rceil) + 1$

$else$

$Num = \lceil \frac{f_{max}}{\alpha} \rceil + j1\mathrm{mod}(\lceil \frac{f}{\alpha} \rceil) - j1\mathrm{mod}(\lceil \frac{f}{\alpha} \rceil)_1$

$endif$

$endif$

$endif$

$if(Num1 \neq 0)$

$Num1 = Num1 - 1$

$k = k - 1$

$endif$

Край

Нека $l = 1$. От алокиращата функция следва, че стойностите $u[i, j]$ и $u[i, j - f((i + 1)]$ се изчисляват в една и съща клетка на двумерния правоъгълен систолически масив. Ще разгледаме изчисляването на стойността $u[m, j]$ за $j = 1, \ldots, c$.

В момент $\left\lceil \frac{j}{\lceil \frac{f_{min}}{\alpha} \rceil} \right\rceil$ в клетка $alloc(1, j) = (i, k(1, j))$,

$$k(1,j) = \begin{cases} j \bmod \left\lceil \frac{f(2)}{\alpha} \right\rceil & \text{за } j \bmod \left\lceil \frac{f(2)}{\alpha} \right\rceil \neq 0 \\ \left\lceil \frac{f(2)}{\alpha} \right\rceil & \text{за } j \bmod \left\lceil \frac{f(2)}{\alpha} \right\rceil = 0 \end{cases},$$

постъпва стойността $u[0,j]$ заедно със стойностите $Num = 1$, $Num1 = 0$, j, $j1 = 0$, $u1 = 0$, $u2 = 0$, $u4 = 0$. Ако Num е различно от 0, намаляваме Num с 1, присвояваме на $u1$ стойността $u[i,j]$ (в случая $u[0,j]$). Присвояваме на $Num1$ стойността на Num, на Num присвояваме стойност 0, а на k присвояваме стойността $\lceil \frac{f_{max}}{\alpha} \rceil - 1$. На $j1$ присвояваме стойността на j. Проверяваме дали $Num1$ е равно на 0. Ако $Num1$ е равно на 0 и $f(i) \geq j1 = j$, тогава $u[i+1,j] = u[i,j]$ (в случая $u[1,j] = u[0.j]$) и записваме тази стойност в регистър с номер $(j \bmod f(i)+1) \bmod \alpha$ на локалната памет. Намаляваме стойността на k с 1. Ако стойността на k е различна от 0, стойността на $u[i+1,j]$ (в случая $u[1,j]$) престоява k момента в регистър $u3$, k е брояч, необходим за синхронизация на данните, преминаващи през масива.

Ако $f(i) < j1$, то изчисляваме стойността $u[i+1,j] = F(u[i,j], u[i, j - f(i+1)] + g(i)]$. Стойността $u4 = u[i, j - f(i+1)]$ влиза в клетката едновременно със стойността $u[i,j]$. След това на променливата $u2$ присвояваме стойността $u[i+1, j - f(i+2)]$ намираща се в регистър $(j \bmod f(i)+1) \bmod \alpha$ на локалната памет. В същия регистър записваме новополучената стойност $u[i+1,j]$. Намаляваме стойността на k с 1. Ако стойността на k е различна от 0, стойността на $u[i+1,j]$ престоява k момента в регистър $u3$ на локалната памет.

Ако $j \bmod \left\lceil \frac{f(i+1)}{\alpha} \right\rceil - j \bmod \left\lceil \frac{f(i+2)}{\alpha} \right\rceil \geq 0$, на променливата Num присвояваме стойността $j \bmod \left\lceil \frac{f(i+1)}{\alpha} \right\rceil - j \bmod \left\lceil \frac{f(i+2)}{\alpha} \right\rceil + 1$. В противен случай на променливата Num присвояваме стойността $\lceil \frac{f_{max}}{\alpha} \rceil + j \bmod \left\lceil \frac{f(i+1)}{\alpha} \right\rceil -$

$j \bmod \left\lceil \frac{f(i+2)}{\alpha} \right\rceil$. Стойността на променливата Num показва през колко клетки трябва да премине стойността $u[i,j]$ за да достигне клетката, където се изчислява стойността $u[i+1,j]$.

Ако $Num1$ е различно от 0 намаляваме стойността на $Num1$ с 1, намаляваме стойността на k с 1 и предаваме данните на съседните клетки, където операциите се повтарят.

Ако Num е равно на 0, предаваме данните на съседните клетки.

Така в момент $t(m,j)$ стойностите $u[m-1,j]$, $Num = 0$, $Num = 1$, $k = 1$, j, $j1 = j$, $u1 = u[m-1,j]$, $u2 = u[m-1, j-f(m)]$, $u4 = u[m-1, j-f(m)]$ влизат в клетка с номер $alloc(m,j)$ и изчисляват стойността $u[m,j]$. Това е една от изходните стойности.

Следната програма, написана на псевдо-програмен език, показва как работи отделната клетка на двумерния правоъгълен систолически масив.

Начало:

$if(Num \neq 0)$

$Num = Num - 1$

$u1 = u$

$u4 = u2$

$Num1 = Num$

$Num = 0$

$k = \left\lceil \frac{f_{max}}{\alpha} \right\rceil - 1$

$j1 = j$

$endif$

$if(Num1 = 0)$

$if(f \geq j1)$

$u = u1$ /записваме стойността на u в регистър с номер $(j \bmod f(i) + 1) \bmod \alpha$ на локалната памет/

$k = k - 1$

$if(k \neq 0)$

/стойността на променливата u престоява k момента в регистър $u3$/

$endif$

$if(f < j1)$

$u = F(u1, u4 + g)$ /на променливата $u2$ присвояваме стойността от регистър с номер $(j \bmod f(i) + 1) \bmod \alpha$ от локалната памет и в същия регистър записваме новополучената стойност u/

$u1 = 0$

$k = k - 1$

$if(k \neq 0)$

/стойността на променливата u престоява k момента в регистър $u3$/

$endif$

$endif$

$if(j1 \bmod(\lceil \frac{f}{\alpha} \rceil) - j1 \bmod(\lceil \frac{f_1}{\alpha} \rceil)) \geq 0)$

$Num = j1 \bmod(\lceil \frac{f}{\alpha} \rceil) - j1 \bmod(\lceil \frac{f_1}{\alpha} \rceil) + 1$

$else$

$Num = \lceil \frac{f_{max}}{\alpha} \rceil + j1 \bmod(\lceil \frac{f}{\alpha} \rceil) - j1 \bmod(\lceil \frac{f_1}{\alpha} \rceil)$

$endif$

$endif$

$if(Num1 \neq 0)$

$Num1 = Num1 - 1$

$k = k - 1$

$endif$

Край.

Теорема 6 *На изхода на така построения двумерен правоъгълен систолически масив се получават необходимите стойности* $u[m, j]$.

Доказателство. От програмата за работа на отделната клетка следва, че в момент $t(i,j)$ на входа на клетка с номер равен на $alloc(i,j)$ постъпва стойност $u = u[i-1,j]$. Стойността $u2 = u[i-l, j-f(i)]$, $l \in \{0,1\}$, е вече изчислена, следва от времевата функция, и се намира в същата клетка в регистър с номер равен на $(j1 \bmod(f)+1) \bmod(\alpha)$, при $l=0$ и влиза едновременно с $u = u[i-1,j]$ при $l=1$. Така в тази клетка се пресмята стойността $u = F(u[i-1,j], u[i-l,j-f(i)]+g)$. От това, че времевата функция е валидна времева функция следва, че на входа на всяка от клетките влизат по една променлива u и $u1$, $u2$ и $u4$ като едната от променливите има стойност 0, а другата има стойност $u[i-1,j]$, за първите две и 0 и $u[i-1, j-f(i)]$, за вторите две, т.е. няма конфликт на данните. От времевата функция получаваме:

$$t(i+1,j) - t(i,j) =$$

$$= i\left\lceil \frac{f_{max}}{\alpha} \right\rceil - \left\lceil \frac{j}{\lceil \frac{f_{min}}{\alpha} \rceil} \right\rceil - (i-1)\left\lceil \frac{f_{max}}{\alpha} \right\rceil + \left\lceil \frac{j}{\lceil \frac{f_{min}}{\alpha} \rceil} \right\rceil = \left\lceil \frac{f_{max}}{\alpha} \right\rceil .$$

От брояча $k = \lceil \frac{f_{max}}{\alpha} \rceil - 1$ получаваме, че стойността $u[i-1,j]$ се получава в клетка $alloc(i,j)$, $\lceil \frac{f_{max}}{\alpha} \rceil$ момента след като бъде изчислена. Следователно, стойността $u[i+1,j]$ се получава в клетка $alloc(i,j)$ в момента в който изчисляваме стойността $u[i,j]$.

Следователно на изхода получаваме необходимите стойности $u[m,j]$.

□

3.3 Двумерен правоъгълен систолически масив с фиксиран брой изчислителни елементи

До сега предполагахме, че разполагаме с толкова клетки, колкото са ни необходими В тази секция разглеждаме двумерен правоъгълен систолически масив, съставен от $n \times q$ изчислителни елемента (клетки). Без ограничение на общността допускаме, че $n \leq m$ и $q \leq \lceil \frac{f_{max}}{\alpha} \rceil$, ако $n > m$ или $q > \lceil \frac{f_{max}}{\alpha} \rceil$ клетките от стълбовете от $m+1$ до n и от редовете от $\lceil \frac{f_{max}}{\alpha} \rceil + 1$ до q не се използват за изчисляване на стойностите $u[i,j]$. В случая, двумерният правоъгълен систолически масив е съставен от $n \times q$ клетки от същият тип, както в предходната секция. Всеки стълб има опашка с размер $\lceil \frac{f_{max}}{\alpha} \rceil - q$, която представлява памет от тип $FIFO$ (първият влязъл е първият излязъл) и се използва за прехвърляне на данни от последната клетка на стълба към първата клетка. Всеки ред има опашка с размер $m - n$, която представлява памет от тип $FIFO$ и се използва за прехвърляне на данни от последната клетка на реда към първата клетка.

Разделяме стойностите $u[i,j]$, $i = 1, \ldots, m$, $j = 1, \ldots c$ на $\lceil \frac{m}{n} \rceil \left\lceil \frac{\lceil \frac{f_{max}}{\alpha} \rceil}{q} \right\rceil$ множества B_{dr}, които се обработват последователно в нарастващ ред първо спрямо q, а след това спрямо n и $u[i,j] \in B_{dr}$, $d = 1, \ldots, \lceil \frac{m}{n} \rceil$, $r = 1, \ldots, \left\lceil \frac{\lceil \frac{f_{max}}{\alpha} \rceil}{q} \right\rceil$, точно тогава когато $\lceil \frac{i}{n} \rceil = d$ и $\left\lceil \frac{j \bmod (\lceil \frac{f_{max}}{\alpha} \rceil) + 1}{q} \right\rceil = r$. Всяко множество B_{dr}, $d = 1, \ldots, \lceil \frac{m}{n} \rceil$, $r = 1, \ldots \left\lceil \frac{\lceil \frac{f_{max}}{\alpha} \rceil}{q} \right\rceil$ се реализира върху тора от изчислителни елементи, както беше разгледано в предходната секция, т.е. той се реализира с времева сложност:

$$t_{dr} = nq + \left\lceil \frac{c}{\min\{q, \lceil \frac{f_{max}}{\alpha} \rceil\}} \right\rceil .$$

Множествата B_{dr} са $\lceil \frac{m}{n} \rceil \left\lceil \frac{1}{q} \lceil \frac{f_{max}}{\alpha} \rceil \right\rceil$ на брой и се реализират последователно, следователно времевата сложност на цялата задача върху $n \times q$ изчислителни елемента е:

$$T = \left\lceil \frac{m}{n} \right\rceil \left\lceil \frac{\lceil \frac{f_{max}}{\alpha} \rceil}{q} \right\rceil \left\lceil \frac{c}{\min\{q, \lceil \frac{f_{min}}{\alpha} \rceil\}} \right\rceil + nq .$$

Нека сравним линейната и двумерната реализации на задачата, когато двата систолически масива съдържат равен брой клетки $p = nq$. Времевата сложност на задачите описани с уравнение (2.1), реализирани върху двумерен правоъгълен систолически масив с $p = nq$ изчислителни елемента е:

$$T_2 = nq + \left\lceil \frac{c}{\min\{q, \lceil \frac{f_{min}}{\alpha} \rceil\}} \right\rceil \cdot \left\lceil \frac{m}{n} \right\rceil \cdot \left\lceil \frac{\lceil \frac{f_{max}}{\alpha} \rceil}{q} \right\rceil .$$

Времевата сложност на задачите описани с уравнение (2.1), реализирани върху линеен систолически масив с $p = nq$ изчислителни елемента е:

$$T_1 = p + c \left\lceil \frac{\sum_{i=1}^{m} w_i}{p} \right\rceil \leq p + \frac{c}{p} \sum_{i=1}^{m} \frac{f(i)}{\alpha} .$$

Коя от двете реализации е по-добра зависи от това, коя от стойностите $\sum_{i=1}^{m} f(i)$ и $\frac{m f_{max}}{\min\{q, \frac{f_{min}}{\alpha}\}}$ е по-малка. Така при стойности на $f(i)$, $i = 1, \ldots, m$, близки до f_{min} $T_1 < T_2$, а при стойности на $f(i)$, близки до f_{max} $T_1 > T_2$.

Пример 2 *Нека разполагаме с линеен и с двумерен правоъгълен систолически масиви, съдържащи равен брой изчислителни елементи* $p = nq$, *където* $n = 2$ $q = 3$. *Нека локалната памет на всяка от клетките да бъде* $\alpha = 1$ *регистъра. Нека* $m = 4$, $c = 5$. *При* $f(1) = 1$, $f(2) = 2$, $f(3) = 3$, $f(4) = 4$, *за времевата стойност на линейната реализация получаваме* $T_1 = 16$, *а за времевата сложност на двумерната реализация получаваме* $T_2 = 26$. *Докато при същите стойности за* m, c, n, *и* q *и* $f(1) = 2$, $f(2) = 3$, $f(3) = 3$, $f(4) = 3$ *получаваме, че* $T_1 = 16$, *а* $T_2 = 10$. □

Така ако разполагаме с някаква архитектура, например хиперкуб, можем да преценим, как е по-добре да реализираме задачата. В едни случаи използването на алгоритъма за линеен систолически масив е по-подходящо, а в други случаи по-подходящо е използването на алгоритъма за двумерен правоъгълен систолически масив.

3.4 Заключение

В тази глава разгледахме един клас от нехомогенни рекурентни уравнения, породени от важни практически задачи. Дефинирахме съответстващият им граф на зависимостите, който се определя от неконстантните входни данни $f(i)$, $i = 1, \ldots, m$. Разгледахме разполагането на тази задача върху двумерен правоъгълен систолически масив. Доказахме, че на изхода на систолическия масив се получават необходимите стойности. Коя реализация е по-добра, от гледна точка на времевата сложност, линейната или двумерната, зависи от стойностите $f(i)$, $i = 1, \ldots, m$. Двумерният правоъгълен систолически масив има Хамилтонов път (път минаващ през всички клетки по веднъж). Така, ако потребителят разполага с двумерен правоъгълен систолически масив, в зависимост от

стойностите $f(i)$, $i = 1, \ldots, m$, той може да прецени коя реализация е по-подходяща, линейната или двумерната. Ако разгледаме случая, когато $\alpha = 1$, n дели m, q дели f_{max} получаваме, че времевата сложност на реализацията върху двумерен правоъгълен систолически масив е:

$$T = \frac{mcf_{max}}{n.q \min\{q, f_{min}\}} + nq.$$

При $q = 1$ или $f_{min} = 1$ нашият резултат се изравнява с резултата на Andonov и Quinton, в останалите случаи нашият резултат е по-добър.

Глава 4

Разширими систолически масиви

Систолическите масиви предложени от H.T. Kung и E. Leiserson [32], са интересна и много популярна паралелна компютърна архитектура, много подходяща за директно разполагане върху $VLSI$ (свръх големи интегрални схеми). Най-важните им качества са регулярност на комуникационната им структура, постоянни потоци от данни, минаващи през систолическия масив, многократно използване на въведените в масива входни данни. Очевидните предимства на систолическите масиви [30] водят до голям интерес към тях и постигане на разнообразни резултати. Едно от направленията по които се работи е конструиране на систолически масиви за специфични задачи: умножение на матрици, решаване на системи линейни уравнения и др. [1], [31], [32] . Изследователите използват своя опит и интуиция за конструиране на различни систолически масиви, докато въпросът за тяхната оптималност и коректност се изследва след това. Този подход използваме в глави 2 и 3 на дисертацията.

Друго направление е свързано със задачата да се открие методология за автоматично трансформиране на алгоритмите върху систолически масиви, които да реализират тези алгоритми коректно. В пре-

дишните глави показахме реализацията на един конкретен алгоритъм върху линеен и двумерен систолически масиви. Стана ясно с какви трудности се сблъскваме. Дотук ставаше дума за един сравнително прост систолически масив.

При по-сложни масиви и особено, когато се състоят от повече от един вид клетки, задачата за реализация на отделен проблем става изключително трудна. За това много актуален е въпросът за автоматизация на процеса на проектиране. Тази задача се решава по различен начин: трансформации върху системи рекурентни уравнения [17], [27], [46] и оптимизиране [41], геометрични трансформации [54], [55], алгебрични трансформации [33] и др. .

В тази глава е демонстриран подход, който се основава на идеята да изследваме директно не алгоритъма и изчислителната структура, а структури, които са значително по-малки и все пак запазват основните свойства на изходната архитектура. Малкият размер на тези структури прави възможно разглеждането на голям брой кандидати за изобразяване и така можем да намерим най-доброто решение по някакъв критерий. Автоматичният синтез води до улесняване на проектирането, а от там до поевтиняване на тяхната промишлена реализация. Теорията на графите се използва за описание и на алгоритъма и на систолическия масив. Теоретичните резултати лесно могат да се използват за конструиране на систолически масив с желаните свойства.

В началото ще дадем някои основни дефиниции и означения , необходими в процеса на изложението.

Дефиниция 5 *[19] Поточен граф се нарича ацикличен граф с цветове на върховете и цветове и етикети на ребрата, удовлетворяващ условията:*

1. Ребра с еднакъв етикет имат еднакъв цвят.

2. За всеки етикет s съществува път започващ и завършващ във върхове от степен 1 и съдържащ всички ребра с етикет s и само тях.

3. За всеки цвят c и за всеки връх v съществува най-много едно ребро с цвят c влизащо/излизащо от v.

4. За всяка двойка от върхове v, w, всички пътища между връх v и връх w имат еднаква дължина.

Нека G е поточен граф. Можем да въведем релация на частична наредба " $\leq$ ". $v \leq w$, ако съществува път от v до w. Минималните върхове по отношение на тази релация ще наричаме *входни*, а максималните - *изходни*. Всички входни и изходни върхове образуват множеството на *терминалните* върхове. Ребрата инцидентни с тях ще наричаме *терминални ребра* (входни и изходни).

От условие 4 на дефиниция 5 следва, че е възможно да определим неотрицателно ниво $lev(.)$ на нетерминалните върхове на поточния граф G така, че за всяко ребро $(v, w) \in V(G)$, $lev(w) = lev(v) + 1$ и да съществува връх $z \in V(G)$ така,че $lev(z) = 0$.

От дефиниция 5 следва лемата:

Лема 1 *Ако ребро с цвят c влиза в/излиза от нетерминален връх v на поточния граф G, то точно едно ребро с цвят c излиза от/влиза в v.*

Ще използваме следните означения, ако $v, w \in V(G)$ и $e = (v, w) \in E(G)$, то с $col(v)$ ще се означава цвят на v, с $col(e)$ - цвят на e, а с $lab(e) = lab((v, w))$ - етикет на e. С $E_n(G)$ ще бележим множеството на нетерминалните ребра на граф G, а с $V_n(G)$ - множеството от терминални върхове на G.

Когато поточен граф се използва за описание на даден алгоритъм, върховете му съответстват на изчисленията, а ребрата на зависимостта

между тях. Цветовете на ребрата съответстват на различните потоци данни, а етикетите на придвижването на една информационна единица. Цветовете на върховете съответстват на различните видове изчисления. В този случай поточният граф е граф на зависимостите.

Пример 3 *Ще разгледаме алгоритъм за намиране на произведението* C *на две матрици* A *и* B *с размер* $n \times n$. *Всяко пресмятане може да се представи във вида:*

С $C(i,j,k)$ *бележим* k*-тото междинно пресмятане за получаване на елемента* $c(i,j)$. *Към елементите на матрици* A *и* B *се добавя по един допълнителен индекс и така множествата от данни участващи в алгоритъма са с еднаква размерност.*

$C(i,j,k) = C(i,j,k-1) + A(i,k,j-1) \times B(k,j,i-1)$

$A(i,k,0) = a_{ik}$ - *елемент на* A

$B(k,j,0) = b_{kj}$ - *елемент на* B

$C(i,j,0) = 0$

$A(i,k,j) = A(i,k,j-1)$

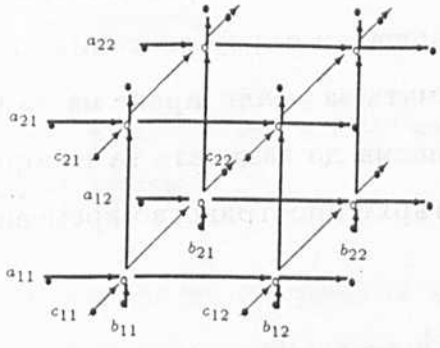
$B(k,j,i) = B(k,j,i-1)$

$C(i,j,N) = c_{ij}$ - *елемент на* C

$1 \leq i \leq N \quad 1 \leq j \leq N \quad 1 \leq k \leq N$

Графът на зависимостите, отговарящ на този алгоритъм за $N = 2$ *е даден на фиг.4.1.*

Когато използваме поточен граф за изобразяване на систолически масив, върховете съответстват на изчислителните елементи, а ребрата на физическата връзка между тях. Цветовете и етикетите на ребрата показват придвижването на информацията, а цветовете на върховете съответстват на различните видове изчислителни елементи. В този случай, ще наричаме поточния граф *хардуерен* граф.



Фигура 4.1: Граф на зависимостите на алгоритъма за умножение на (2×2) матрици

Всеки поточен граф може да се разглежда като граф на зависимостите или хардуерен в зависимост от интерпретацията.

Ще използваме изобразяване на поточен граф на зависимостите върху хардуерен граф за представяне на реализацията на даден алгоритъм върху даден систолически масив.

Дефиниция 6 *[19]Нека $H = (V, E)$ е хардуерен граф. Означаваме:*

$V' = V \times \{0, 1, 2, ...\}$

$col((v, t)) = col(v) \quad v \in V, \quad t \in 1, 2, ... \quad (v, t) \in V'$

$E' = \{(v', w') : \quad v' = (v, t_1) \in V', \quad w' = (w, t_2) \in V', \quad v, w \in V, \quad (v, w) \in E, t_2 = t_1 + 1\}$

$col((v', w')) = col((v, w))$

Граф $G' = (V', E')$ ще наричаме пространство-времеви граф на H.

Ако $v' = (v, t) \in V'$, то ще наричаме върха v *проекция* на върха v' и ще го бележим с $pr(v')$, а t - е равно на нивото $lev(v')$ на v'.

Чрез пространство-времевия граф изобразяваме безкрайна редица от изчисления извършвани върху систолическия масив.

Ще сведем задачата за реализиране на даден алгоритъм върху даден систолически масив до задачата за изобразяване на поточен граф на зависимостите върху пространство-времевия граф на даден хардуерен граф.

Дефиниция 7 *[19].Нека G е поточен граф на зависимостите, а G' - пространство-времеви граф на някой хардуерен граф H. Систолическа реализация на G върху G' ще се нарича изображение $M : G \to G'$, удовлетворяващо следните условия:*

1. Ако $v, w \in V(G)$, $v \leq w$, то $M(v) \leq M(w)$.

2. Ако $(v, w) \in E(G)$, то $(M(v), M(w)) \in E(G')$. От това условие M дефинира изображение на $E(G)$ във $E(G')$, което също ще отбелязваме с M, така $M((v, w)) = (M(v), M(w))$.

3. Ако $v \in V(G)$, $e \in E(G)$, то $col(M(v)) = col(v), col(M(e)) = col(e)$, т.е. M запазва цветовете.

4. Ако v е терминален връх на G, то $pr(M(v))$ е терминален връх на H.

5. Ако $(v', w') \in E(G')$ и съществува $v \in V(G)$, такова че $v' = M(v)$, то съществува $(v, w) \in E(G)$, такова че $w' = M(w)$.

Дефиниция 8 *Нека G е поточен граф на зависимостите, а H е хардуерен граф. Нека M е систолическа реализация на поточния граф на зависимостите G във пространство-времевия граф на H. Тогава времето на M е $T = max\{lev(M(v)) : v \in V(G)\}$.*

Изображението M може да се представи като двойка изображения (S, T), където за всяко $v \in V(G)$, $S(v) = pr(M(v))$, $T(v) = lev(M(v))$.

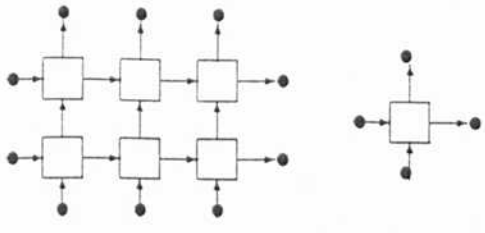
Дефиниция 9 *Регулярен път с етикет s се нарича произволен път в G, чиито ребра имат един и същ етикет s.*

Дефиниция 10 *Максимален регулярен път в G се нарича регулярен път, свързващ двойка терминални върхове.*

Нека M е систолическа реализация на графа на зависимостите G във пространство-времевия граф на хардуерния граф H. Изображението M се дели на двойка изображения $M = (S, T)$, за които от дефиниции 2 и 1 се получава, че са съответно алокиращa и времева функции.

4.1 Хомогенни систолически масиви

В тази секция е разгледан метод за синтез на систолически масиви, състоящи се само от един вид изчислителни елементи. Тези масиви ще наричаме хомогенни. Този метод използва правилността на комуникационната мрежа. Части от този проблем са решени в [18], [19], но там са разгледани само хомогенни систолически масиви с правоъгълна структура. Систолически масив за даден алгоритъм, може да се получи от систолически масив за алгоритъм от същия тип, но с по-малък размер. Това се постига, като се намери изображението между алгоритъм и систолически масив за алгоритъм с малък размер, а след това изображението се разшири до необходимият размер. Новото в този метод е използването на допълнителни върхове, като по този начин се разширява класа от систолически масиви които методът обхваща, за разлика от [19] и [18]. Изведено е изображение на алгоритъма, представен чрез граф, върху систолически масив.



Фигура 4.2: Редукция на граф

Цел на тази работа е да се открие начин за изобразяване на един поточен граф на зависимостите върху пространство-времевия граф на даден хардуерен граф.

Ефективна систолическа реализация могат да имат само алгоритми с правилна структура. Такива алгоритми могат да бъдат представени, използвайки понятието рекурсивни редици от графи, което ще бъде дефинирано по-долу.

Дефиниция 11 *[19] Нека G е свързан поточен граф. Редукция на G ще наричаме замяната на всички негови не терминални върхове с един (редуцирано ядро на G) и на всички едноцветни терминални ребра с едно от същия цвят (фиг.4.2)*

В тази част е разгледано разширяването на даден поточен граф (граф на зависимостите или хардуерен). Разширеният граф съответства на граф на зависимостите или на хардуерен граф, за същия алгоритъм, но с по-голям размер. От разширенията на даден граф, образуваме редица наречена **рекурсивна редица**. След това правим разширение на систолическата реализация на графа на зависимостите върху хардуерния граф.

От дефиницията за редукция на поточен граф следва, че е в сила следната лема:

Лема 2 *Необходимо и достатъчно условие да може да бъде извършена редукция на поточния граф G е за всеки връх v и за всеки цвят с от граф G_0 да съществува ребро от цвят с което влиза/излиза от v.*

.

За Всеки поточен граф G ще дефинираме множество от ребра $E'(G) = \{(v,w) :$ съществува максимален регулярен път $(x_1,\ldots,x_k)$ в G така,че $v = x_{k-1}$, а $w = x_2\}$. Цветът на ребрата в множеството $E'(G)$ ще е същият като цвета на ребрата съставящи максималния регулярен път.

Нека е даден поточният граф G_0, за който може да се извърши редукция. Нека на мястото на всеки връх от G_0, се постави граф изоморфен на G_0. Ако $(x,y) \in E(G_0)$, то премахваме изходните терминални ребра от цвят $col(x,y)$ за графа заменящ върха x и входните терминални ребра от цвят $col(x,y)$ за графа заменящ върха y. Реброто (x,y) заменяме с път от същият цвят и посока. Нека x и y са два съседни върха в граф G_0. С $[x,v]$ ще бележим върхът, получен при замяната на връх x в граф G_0 с граф изоморфен на граф G_0, а в заменящия граф това е връх v. Нека върховете $v,w \in G_0$, и $(w,v) \in E'(G_0)$, $col(w,v) = 1$. Нека $(x,y) \in E(G)$ и $col(x,y) = 1$, тогава пътят от цвят 1 свързващ върхове $[x,v]$, $[y,v]$ и пътят от цвят 1 свързващ върхове $[x,u]$, $[y,u]$ ще искаме да са с равна дължина, u е произволен връх от G_0. Ако v и w са инцидентни с терминални ребра от един и същи цвят и тези ребра са едновременно входни или едновременно изходни и $(v,w) \in E(G_0)$ $(x,y) \in E(G_0)$ и също са съседни на терминални ребра, тогава $([x,v],[y,w])$ е ребро от новополучения граф, т.е. не се поставят допълнителни върхове в съседство с терминални върхове. Така получаваме граф G_1, който ще наричаме разширение на G_0 с граф G_0. Ако върховете на G_1 се заменят с графи изоморфни на G_0, по същия начин както по-горе, то ще получим разширение G_2 на G_1 с G_0 и т.н. .Получаваме редица от графи, която ще

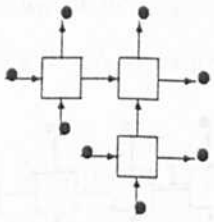
наричаме рекурсивна редица от графи, породена от G_0, а G_0 - пораждащ граф. Поточен граф G_0, за който може да се построи рекурсивна редица, ще наричаме разширим.

Лема 3 *Допълнителните върхове, поставени за път от даден цвят, са допълнителни и за пътищата от другите цветове.*

Доказателство. Нека е даден граф G_0. От описания по-горе алгоритъм получаваме граф G_1. Нека x и y са два съседни върха от граф G_0, свързани с ребро от цвят 1, а y и z, както и x и z_1 са съседни върхове, свързани с ребра от цвят 2. При разширението върховете x, y, z, z_1 се заменят с графи изоморфни на G_0, а свързващите ги ребра заменяме с едноцветни пътища, както вече описахме по-горе. Нека реброто $(x, y) \in E(G_0)$ бъде заменено в G_1 с пътища от същият цвят. За удобство, ще въведем следното означение, $[x, y]$ ще означава връх от граф G_1, получен при замяната на връх $x \in V(G_0)$ с граф G_0 и в самия G_0 е връх y. От алгоритъма за разширяване на поточни графи следва, че дължината на пътя от цвят 1 между върховете $[x, x]$ и $[y, x]$ е равна на дължината на пътя от цвят 1 между върховете $[x, y]$ и $[y, y]$ е равна на дължината на пътя от цвят 1 между върховете $[x, z]$ и $[y, z]$.

От условие 4 на дефиниция 5 за поточен граф имаме, че всички пътища от върха x до върха z имат еднаква дължина. Един от тези пътища е пътят (x, y, z) и има дължина 2. Следователно $(z_1, z) \in E(G_0)$ и $col(z_1, z) = 1$.

Нека по пътя между върховете $[x, x]$ и $[y, x]$ от граф G_1 да няма допълнителни върхове, а по пътя между върховете $[x, z]$ и $[y, z]$ да има един допълнителен връх. Ще означим този допълнителен връх с v. Нека пътят от връх $[x, y]$ до връх $[y, y]$ е съставен от върховете: $[x, y]$, $[x, y_1]$, $[x, y_2]$, ..., $[x, y_n]$, $[y, x_1]$, $[y, x_2]$, ..., $[y, x_m]$, $[y, x]$, $[y, y]$, (максималният



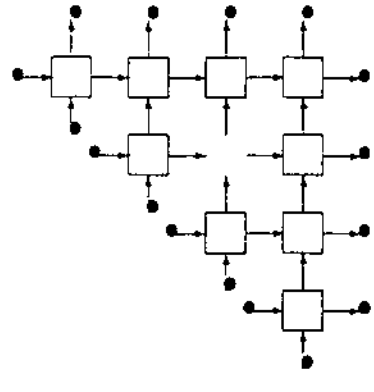
Фигура 4.3:

регулярен път от цвят 1 в граф G, съдържащ върховете x и y е съставен от нетерминалните върхове $x_1, x_2, \ldots, x_m, x, y, y_1, y_2, \ldots y_n)$. Съответно пътят от връх $[x, z_1]$ до връх $[y, z_1]$ е съставен от върховете $[x, z_1]$, $[x, z]$, $[x, z_2]$, $\ldots$, $[x, z_{n+m-k}]$, v, $[y, z_{n+m-k+1}]$, $\ldots$, $[y, z_{n+m}]$, $[y, z_1]$ и максималният регулярен път от цвят 1 в граф G_0 съдържащ върховете z_1 и z е $z_{n+m-k+1}, \ldots, z_{n+m}, z_1, z, z_2, \ldots, z_{n+m-k}$. Така дължината на максималният регулярен път от граф G_0 съдържащ върховете x и y е с 1 по-голям от дължината на максималния регулярен път, съдържащ върховете z и z_1. Следователно в граф G_0 от връх x_{m-k} при $k < m$ ($y_{n+m-k+1}$ при $k > m$) излиза терминално ребро от цвят 2 и максималният регулярен път от граф G_0 от цвят 2 съдържащ връх x_{n-k} ($y_{n+m-k+1}$) е с по-малка дължина, от максималният регулярен път, съдържащ връх y. Следователно връх $[y, x_{n-k}]$ ($[x, y_{n+m-k+1}]$) е съседен на допълнителен връх. следователно връх v е допълнителен и за пътя от цвят 2. □

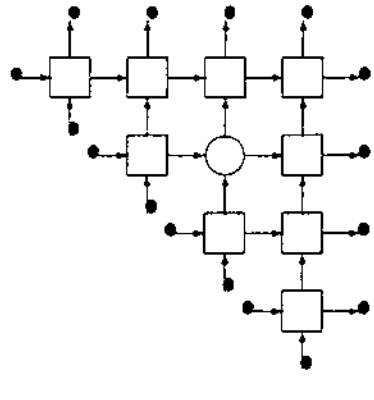
Пример 4 .*Нека е даден поточен граф G_0. Да се построи разширението му G_1 (фиг.4.3).*

1.Заменяме всеки връх от G_0 с G_0 (фиг.4.4)

2. На мястото на свързващите ребра, поставяме пътища от същият цвят и посока (фиг.4.5) с ◯ сме означили допълнителните върхове.



Фигура 4.4:



Фигура 4.5:

Лема 4 *Нека $G_0, G_1, \ldots$ е рекурсивна редица от графи, породена от поточния граф G_0. Тогава за всяко $i > 0$ G_i е също поточен граф.*

Доказателство. Нека $i = 1$. Условия 1, 2 и 3 на дефиниция 5 следват от начина на построяване на разширението. От това, че G_0 е поточен граф следва, че условие 4 на дефиниция 5 е изпълнено за върхове $[x, y]$ и $[x, z]$ от G_1, $x, y, z \in V_n(G_0)$. От изискването едноцветният път между върховете $[v, x]$ и $[w, x]$ да има еднаква дължина с пътя от същия цвят между върховете $[v, y]$ и $[w, y]$, $(v, w) \in E(G_0)$, следва че условие 4 на дефиниция 5 е изпълнено за върхове $[v, x]$ и $[w, y]$.

По индукция ще докажем, че за всяко $i > 1$ граф G_i е поточен граф. Нека допуснем, че G_{i-1} е поточен граф. По условие G_0 също е поточен граф. Следователно във всеки един от върховете на G_0 и на G_{i-1} влиза /излиза най-много едно ребро от цвят c. Следователно и във всеки един от върховете на G_i ще влиза/излиза най-много едно ребро от цвят c, или изпълнено е условие 3 на дефиниция 5. Условия 1 и 2 на дефиниция 5 следват от начина на построяване на разширението. От изискването едноцветният път между върхове $[v, x]$ и $[w, x]$ да има равна дължина с пътя от същият цвят между върховете $[v, y]$ и $[w, y]$, $v, w \in V_n(G_{i-1})$ $x, y \in V(G_0)$ следва, че условие 4 на дефиниция 5 е изпълнено. Следователно G_i е поточен граф. □

Теорема 7 *Необходимо и достатъчно условие поточният граф G да бъде разширим е :*

1. За всеки цвят c и за всеки връх v, съществува ребро от цвят c инцидентно с v;

2. Нека $(v, w) \in E(G_0)$ и $col(v, w) = c$, то е изпълнено едно от двете условия:

а) Дължината на максималния регулярен път от цвят c_1 съдържащ върха v се различава с 1 от дължината на максималния регулярен път от цвят c_1, съдържащ върха w (c_1 е произволен цвят, различен от c) и единият от максималните регулярни пътища от цвят c_1 инцидентни с терминални ребра от цвят c има само един нетерминален връх;

б) Максималният регулярен път от цвят c_1, чиито върхове са инцидентни на входни терминални ребра от даден цвят има равна дължина с максималния регулярен път от цвят c_1, чиито върхове са инцидентни на изходни терминални ребра от същия цвят.

Доказателство. 1. Нека за граф G_0 са изпълнени условията на теоремата. Тогава от условие 1, ако имаме допълнителен връх, лежащ на път от даден цвят, то този връх е допълнителен и за пътищата от другите цветове (лема 3). От условие 2 следва, че няма да имаме допълнителни върхове инцидентни с терминални ребра.

2. Нека G_0 е разширим поточен граф. Следователно за всеки цвят c и за всеки връх v съществува ребро от цвят c инцидентно с v. В противен случай не можем да извършим замяната на всеки връх от G_0 с графа G_0. Щом граф G_0 е разширим поточен граф, от алгоритъма за разширяване не се допуска поставяне на допълнителни върхове инцидентни с терминални ребра. От друга страна се изисква едноцветният път между върховете $[v,x]$ и $[w,x]$ да е равен по дължина на едноцветния път от същия цвят между върховете $[v,y]$ и $[w,y]$, $v,w,x,y, \in V_n(G_0)$ $(v,w) \in E(G_0)$, тогава е изпълнено условие 2. □

Нека $G_0, G_1, \ldots$ и $H_0, H_1, \ldots$ са съответно рекурсивни редици от графи на зависимостите и хардуерни графи. G_i' е пространство-времевият граф на H_i $(i \geq 0)$ и M_0 е систолическа реализация на G_0 върху G_0'. Рекурсивно дефинираме редица от изображения $M_0, M_1, \ldots$. $M_i = (S_i, T_i)$

по следния начин:

$$S_i(v) = \begin{cases} (S_{i-1}(x), S_0(v_j)), & \text{ако } v = [x, v_j] \text{ не е допълнителен връх за } G_i \\ v_j & \text{- връх в } H_i \text{ така, че да се} \\ & \text{запазва съседството на върховете в } H_i, \\ & \text{ако } v \text{ е допълнителен връх за } G_i \end{cases} \tag{4.1}$$

$$T_i(v) = \begin{cases} (T_{i-1}(x) - 1)(d-1) + T_0(v_j), & \text{ако } v = [x, v_j] \text{ не е допълнителен} \\ & \text{за } G_i,\ x \in V(G_{i-1}),\ v_j \in V(G_0) \\ & d \text{ е най-дългият от всички} \\ & \text{едноцветни пътища.} \\ T_i(w) - 1, & \text{ако } v \text{ е допълнителен и} \\ & (v, w) \in E(G_i) \end{cases} \tag{4.2}$$

В [18] и [19] е разгледано разширяване на графи, при които няма допълнителни върхове. Там аналогично на M_i изображение запазва съседството на върховете. Следователно в разглеждания в тази работа случай, изображението M_i запазва съседството на върховете, които не са допълнителни. От друга страна M_i запазва съседството на допълнителните върхове по дефиниция. Следователно M_i запазва съседството на върховете.

Теорема 8 *Нека $G_0, G_1, \ldots$ е рекурсивна редица от графи на зависимостите, $H_0, H_1, \ldots$ е рекурсивна редица от хардуерни графи, G_i' е пространство-времевият граф на H_i $(i \geq 0)$. $M_0, M_1, \ldots$ е рекурсивна редица от изображения, където M_0 е систолическа реализация на G_0 върху G_0'. Тогава M_i е систолическа реализация на G_i върху G_i' за $i > 0$.*

Доказателство. В [18] е разгледано разширяването на графи, при които няма допълнителни върхове. За този случай там е доказана теорема аналогична на теорема 8, т.е. изображението M_i, за $i = 1, 2, \ldots$, отговаря на условията на дефиниция 7, за върховете, които не са допълнителни. За това теоремата ще бъде доказана само за допълнителните върхове. Допускаме, че изображението M_j, $j = 0, \ldots, i-1$ е систолическа реализация на графа на зависимостите G_j върху пространство-времевия граф на хардуерния граф H_j. Ще докажем, че изображението M_i е систолическа реализация на графа на зависимостите G_i върху пространство-времевия граф на хардуерния граф H_i.Условие 1 на дефиниция 7 е очевидно изпълнено. Нека v е допълнителен връх за граф G_i, от това че връх v се изобразява във връх от граф H_i, при изображението M_i, така че да се запазва съседството на върховете, следва че са изпълнени условия 3 и 5 на дефиниция 7.От алгоритъма за разширяване на графи, не се поставят допълнителни върхове в съседство с терминални върхове. Условие 4 се отнася за терминалните върхове на графите, следователно това условие е изпълнено.

Нека $(v, w) \in E(G_i)$ и v е допълнителен връх, следователно от (4.1) $S_i(v) = v_j$ е връх от H_i запазващ съседството на върховете, т.е. $(S_i(v), S_i(w)) \in E(H_i)$. От (4.2) $T_i(v) = T_i(w) - 1$, следователно $(M_i(v), M_i(w)) \in E(G_i')$. Следователно изображението M_i удовлетворява условие 2 на дефиниция 7.Следователно M_i е систолическа реализация на G_i върху G_i' за $i > 0$. □

4.2 Хетерогенни систолически масиви

В тази част разглеждаме разширяването на хетерогенни систолически масиви. Под хетерогенен систолически масив ще разбираме, систоли-

чески масив, при който има няколко различни вида изчислителни елементи. При разширяването на хетерогенните систолически масиви ще използваме разширяването на хомогенни систолически масиви. По този начин, от систолически масив за дадена задача, може да се получи систолически масив за задача от същия тип, но с по-голям размер. За сега, подобен проблем е разглеждан само за хомогенни систолически масиви (систолически масиви които се състоят само от един вид изчислителни елементи) и при които поточните графи с които се описват имат равни дължини на максималните регулярни пътища от един и същи цвят [18, 19]. В тази секция е разгледан алгоритъм за разширяване на хетерогенен систолически масив и е доказана неговата коректност.

4.2.1 Алгоритъм за разширяване на хетерогенни систолически масиви

Когато за решаването на някаква задача ни се налага да извършваме само един тип изчисления,систолическия масив,на който се реализира тази задача има само един тип клетки т.е. цвета на върховете на графа изобразяващ този систолически масив ще бъде един и същ.

Нека е дадена задача за решаването на която трябва да извършим n различни вида изчисления. Тогава систолическият масив на който се реализира тази задача ще има n типа клетки т.е. върховете на графа изобразяващ този систолически масив ще бъдат оцветени в n различни цвята. Систолически масиви съдържащи различни типове клетки ще наричаме хетерогенни систолически масиви. Целта е да се построи разширение на даден хетерогенен поточен граф. За разлика от хомогенния случай началният граф G_0 няма да бъде поставен на мястото на всеки един от върховете на графа, тъй като имаме различни типове

(цветове) върхове.

Нека G е хетерогенен поточен граф и нека върховете на G са оцветени в s различни цвята. G ще бъде разделен на q графа ($q \geq s$) $D_1, \ldots, D_q$ по следния начин:

1. D_i, $i \in [1, q]$ съдържа нетерминални върхове само от един и същ цвят и свързващите ги ребра;

2. $\cup_{i=1}^{q} V_n(D_i) = V_n(G)$, $\cup_{i=1}^{q} E_n(D_i) \subset E_n(G)$

3. Ако $x \in V_n(D_i)$, $y \in V_n(D_i)$, $i \neq j$ и $(x, y) \in E_n(G)$ то в D_i от върха x излиза терминално ребро с цвят $col(x, y)$, а в D_j влиза терминално ребро с цвят $col(x, y)$.

4.Ако $x \in V_n(D_i)$, $y \in V_n(D_j)$, $(x, y) \in E_n(G)$, $i, j \in [1, q]$ $i \neq j$, тогава D_i и D_j имат равен брой терминални ребра от цвят c.

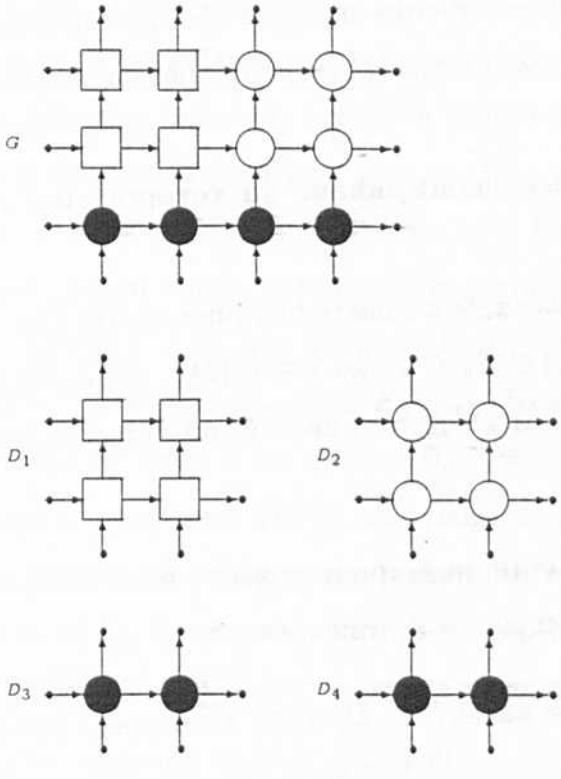
Лема 5 Графите $D_1, \ldots, D_q$ са поточни графи.

Доказателство. Условия 1,3 и 4 на дефиниция 5 за поточни графи са изпълнени, защото $\cup_{i=1}^{q} V_n(D_i) = V_n(G)$ и $\cup_{i=1}^{q} E_n(D_i) \subset E_n(G)$. Условие 2 на дефиниция 5 е изпълнено, защото при построяването на графите D_i, $i = 1, \ldots, q$ там където е премахнато ребро от граф G е поставено терминално ребро от същият цвят имащо същият етикет.□

Пример 5 . *Нека G е поточен граф с върхове оцветени в три цвята 1,2,3 (фиг.4.6).*

Ако се спазват горе описаните условия, от графа G ще получим четири графа D_1, D_2, D_3, D_4, всеки от които представлява хомогенен поточен граф.

В предходната част е описано как се разширява хомогенен поточен граф и как се построява рекурсивна редица от такива графи.



Фигура 4.6: Разделяне на хетерогенен граф на хомогенни графи

Нека е даден хетерогенният поточен граф G_0, от който получаваме q на брой хомогенни поточни графи $D_{01}, \ldots, D_{0q}$. Разширяваме $D_{01}, \ldots, D_{0q}$ както хомогенни поточни графи и получаваме техните разширения $D_{11}, \ldots, D_{1q}$. Тъй като броят на терминалните ребра от цвят c за D_{0i} и D_{0j} е равен, то и броят на терминалните ребра от цвят c за D_{1i} и D_{1j} също ще е равен. Графът получен от свързването на $D_{11}, \ldots, D_{1q}$ означаваме с G_1 и ще наричаме разширение на G_0 с G_0. Хетерогенните поточни графи, за които можем да построим разширение ще наричаме разширими.

Алгоритъм за разширяване на хетерогенни поточни графи

1. Построяваме хомогенните поточни графи $D_1, \ldots, D_q$, така че $\cup_{i=1}^{q} V_n(D_i) = V_n(G_0)$, $\cup_{i=1}^{q} E_n(D_i) \subset E_n(G)$, ако $x \in V_n(D_i)$, $y \in V_n(D_j)$ $i \neq j$ $(x, y) \in E_n(G_0)$, $cal(x, y) = 0$, то D_i и D_j имат равен брой терминални ребра от цвят $c \rightarrow 2$

2. Проверка дали за всички стойности на i, D_i е разширим поточен граф, $i \in [1, q]$, при да $\rightarrow 3$, при не $\rightarrow 5$

3.Разширяване на $D_1, \ldots, D_q$ като хомогенни графи $\rightarrow 4$

4.Свързване на $D_{11}, \ldots, D_{1q}$ в $G_1 \rightarrow 6$

5.Графът не е разширим $\rightarrow 6$

6.Край □

Ако този алгоритъм се приложи няколко пъти, се получава редица от хетерогенни графи, която ще наричаме *рекурсивна редица от хетерогенни графи* породена от граф G, а G - пораждащ граф за редицата.

4.2.2 Коректност на алгоритъма

В тази част се разглеждат подробности на реализацията и коректността на алгоритъма за разширяване на хетерогенни графи.

В началото ще обясним начина на получаване на хомогенните поточни графи $D_1, \ldots, D_q$, от хетерогенния поточен граф G. Ще разгледаме условието, ако $x \in V_n(D_i)$, $y \in V_n(D_j)$ и $(x,y) \in E_n(G)$ и $col(x,y) = c$, то графите D_i и D_j да имат равен брой терминални ребра от цвят c.

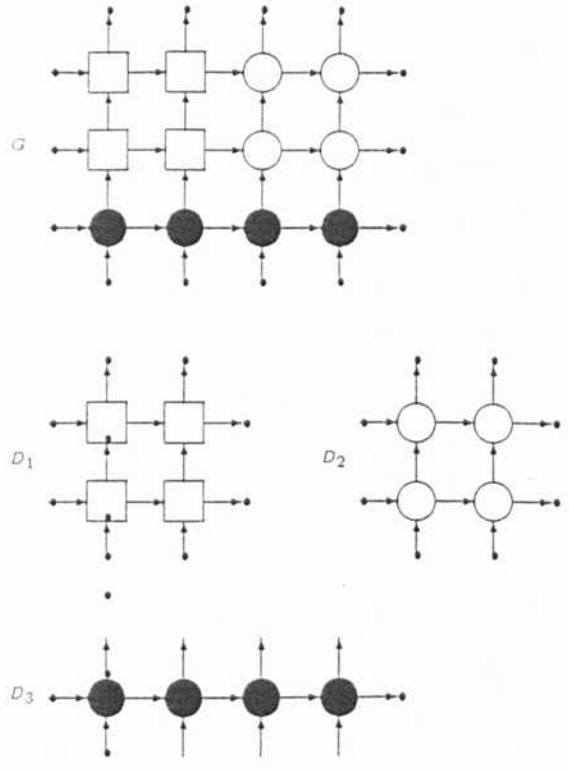
Нека е даден хетерогенен поточен граф G_0 чиито върхове са оцветени в s цвята. Нека построим графите $D_{01}, \ldots, D_{0q}$, като не спазваме условието ако $x \in V_n(D_{0i})$, $y \in V_n(D_{0j})$, $i \neq j$, $(x,y) \in E_n(G_0)$ $col(x,y) = c$, то D_{0i} и D_{0j} да имат равен брой терминални ребра от цвят c. Нека $x_1 \in V_n(D_{0i})$ $y_1 \in V_n(D_{0j_1})$, $i \neq j_1$ и $(x,y) \in E_n(G_0)$ $col(x,y_1) = c, \ldots,$ $x_n \in V_n(D_{0i})$ $y_n \in V_n(D_{0j_n})$, $i \neq j_n$ и $(x_n,y_n) \in E_n(G_0)$. Нека броят на входните терминални ребра от цвят c за графа D_{0j_k}, $k = 1, \ldots, n$ да е n_k, а броят на изходните терминални ребра за графа D_{0i} е $\sum_{i=1}^{n} n_k$. Така след разширяването, броят на входните терминални върхове от цвят c за D_{1j_k}, $k = 1, \ldots, n$ е n_k^2, а броят на изходните терминални ребра от цвят c за графа D_{1i} е $(\sum_{k=1}^{n})^2 > \sum_{k=1}^{n} n_k^2$ и няма как да получим граф G_1.

Пример 6 *Нека е даден поточен граф G (фиг.4.7)*

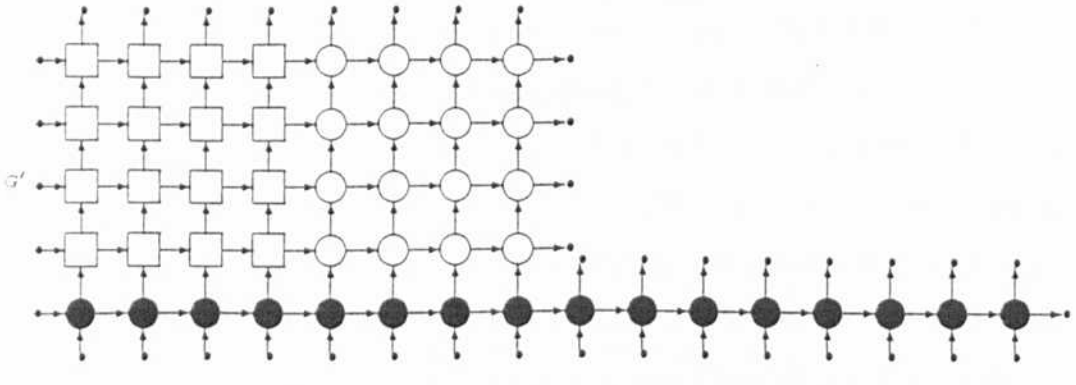
Ако условието броят на терминалните ребра от даден цвят за D_i и D_j да бъде равен, за $i,j \in [1,q]$ $i \neq j$, не е спазено, то от граф G може да получим графите D_1, D_2, D_3.

Нека разширим поточните графи D_1, D_2, D_3 (фиг.4.8).

Получаваме, че броят на терминалните ребра на D_{13}, които трябва да бъдат свързани с терминални ребра от същия цвят на D_{11} и D_{12} е по-голям от броя на терминалните ребра на D_{11} и D_{12} и няма как да получим граф G_1 разширение на G с G.



Фигура 4.7:



Фигура 4.8:

Лема 6 *Нека е дадена рекурсивна редица от хетерогенни поточни графи* $G_0, G_1, \ldots$, *за която пораждащият граф* G_0 *е поточен граф. Тогава за всяко* $i > 0$ G_i *също е поточен граф.*

Доказателство. По условие G_0 е хетерогенен поточен граф. От него получаваме хомогенните поточни графи $D_{01}, \ldots, D_{0q}$, следователно за тях са в сила условията на дефиниция 5. Разширяваме $D_{01}, \ldots, D_{0q}$ като хомогенни поточни графи и получаваме графите $D_{11}, \ldots, D_{1q}$, които също са поточни. От това, че $D_{11}, \ldots, D_{1q}$ са поточни графи следва, че условия 1, 3 и 4 на дефиниция 5 са изпълнени за граф G_1. Остава да проверим изпълнено ли е условие 2 на дефиниция 5 за граф G_1. Изпълнението на условие 2 означава дали за всеки етикет r съществува път започващ и завършващ във върхове от степен 1 (терминални върхове). Нека бъде разгледан път с етикет r в D_{1i} $i \in [1, q]$. Този път започва и завършва с терминален връх в графа D_{1i} защото D_{1i} е поточен граф. Ако допуснем, че пътя с етикет r не завършва с терминален връх в графа G_1, то следователно завършва с връх $x_1 \in V_n(G_1)$. От алгоритъма за получаване на графите D_{0i} $i = 1, \ldots, q$ имаме, че ако $x_0 \in V_n(D_{0i})$, $y_0 \in V_n(D_{0j})$ и $(x_0, y_0) \in E_n(G_0)$, $col(x_0, y_0) = c$ то D_{0i} и D_{0j} имат равен брой терминални ребра от цвят c. Следователно D_{1i} и D_{1j} също ще имат равен брой терминални ребра от този цвят. Следователно всяко изходно терминално ребро на D_{1i} от даден цвят отговаря на входно терминално ребро на D_{1j} от същия цвят. Тогава пътя с етикет r продължава и през D_{1j}. Така условие 2 на дефиниция 5 е изпълнено за графа G_1. Следователно G_1 също е поточен граф. По индукция ще докажем, че графа G_i е поточен граф за $i > 0$. Нека допуснем, че G_{i-1} е поточен граф. По условие G_0 е поточен граф. Следователно от графа G_0 получаваме хомогенните поточни графи $D_{01}, \ldots, D_{0q}$, а G_{i-1} се дели

на хомогенни поточни графи $D_{i-1,1},\ldots,D_{i-1,q}$. От теорема 4 следва, че графите $D_{i1},\ldots,D_{iq}$ също ще бъдат поточни. Следователно условия 1, 3 и 4 на дефиниция 5 са изпълнени и за граф G_i. Остава да се докаже условие 2 на дефиниция 5 . Изпълнението на условие 2 означава за всеки етикет r да съществува път започващ и завършващ във връх от степен 1. Нека бъде разгледан път с етикет r в D_{ij} $j \in [1,q]$. Този път започва и завършва с терминален връх в D_{ij}, защото този граф е поточен. Ако допуснем, че пътя с етикет r не завършва във връх от степен 1 в G_i, то този път завършва с връх $x_i \in V_n(D_{ij})$, $i \in [1,q]$. Тъй като всяко терминално ребро на D_{ij} от даден цвят съответства на терминално ребро на D_{ik} от същия цвят. Следователно пътя с етикет r продължава и през D_{ik}, т.е. условие 2 на дефиниция 5 е изпълнено. □

Друг въпрос, който трябва да разгледаме е, при какви условия хетерогенният поточен граф G е разширим.

Теорема 9 *Необходимо и достатъчно условие хетерогенният поточен граф G_0 да е разширим е, получените от него хомогенни поточни графи да бъдат разширими.*

Доказателство. **Необходимост.** Нека хетерогенният поточен граф G_0 да е разширим. Следователно можем да построим разширение G_1. Нека $G_{01},\ldots,G_{0q}$ са хомогенни поточни графи, получени от графа G_0. Допускаме, че граф G_{0i} не е разширим. В такъв случай не може да се построи негово разширение, а от там и разширение на граф G_0. Следователно граф G_0 не е разширим. Стигаме до противоречие с условието за разширимост на G_0. Това означава, че графите $G_{01},\ldots,G_{0q}$ са разширими хомогенни поточни графи.

Достатъчност. Нека хомогенните поточни графи $G_{01},\ldots,G_{0q}$ получени от поточния хетерогенен граф G_0 построени по условията на алгори-

тъма са разширими поточни графи. Следователно можем да построим техните разширения $G_{11}, \ldots, G_{1q}$. Тези разширения са построени по условията на алгоритъма и могат да се свържат в граф G_1, но този граф ние наричаме разширение на G_0, следователно граф G_0 е разширим поточен граф.□

Забележка. Ако графите $G_{01}, \ldots, G_{0q}$ се състоят от един връх, то разширението съвпада със самия граф.

Нека $G_0, G_1, \ldots$ и $H_0, H_1, \ldots$ са съответно рекурсивни редици от хетерогенни поточни графи на зависимостите и хардуерни графи. Граф G_i' е пространство-времевият граф на H_i $(i > 0)$ и M_0 е систолическа реализация на поточния граф на зависимостите G_0 върху пространство-времевия граф G_0'. Рекурсивно дефинираме редица от изображения $M_0, M_1, \ldots$ $M_i = (S_i, T_i) : V(G) \to V(G')$ по следния начин:

$$S_i(v) = \begin{cases} (S_{i-1}(x), S_0(v_j)), & \text{ако } v = [x, v_j] \text{ не е допълнителен за } G_i \\ v_j & -\text{допълнителен връх в } H_i \text{ ,такъв че} \\ & \text{да се запазва съседството на върховете,} \\ & \text{ако } v \text{ е допълнителен връх за } G_i \end{cases} \tag{4.3}$$

$$T_i(v) = \begin{cases} (T_{i-1}(x) - 1)(d-1) + T_0(v_j), & \text{ако } v = [x, v_j] \text{ не е допълнителен за} \\ & G_i \ x \in V(G_{i-1}),\ v_j \in V(G_0), \\ & d \text{ е най-дългият едноцветен път} \\ & \text{започващ и завършващ} \\ & \text{във върхове от степен 1} \\ T_i(w) - 1, & \text{ако } v \text{ е допълнителен и} \\ & (v, w) \in E(G_i) \end{cases} \tag{4.4}$$

Теорема 10 *Изображението* M_i $(i > 0)$ *дефинирано чрез равенства (4.3) и (4.4) е систолическа реализация.*

Доказателство. Теоремата ще бъде доказана по индукция. В предходната част е доказано, че изображението M_i е систолическа реализация, когато G_i и H_i се състоят от върхове, оцветени в един и същи цвят, т.е. за хомогенни поточни графи. От G_i получаваме q на брой хомогенни поточни графа $D_{i1}, \ldots, D_{iq}$. За тях M_i е систолическа реализация. От това следва, че условия 1,3,5 на дефиниция 7 са изпълнени. Условие 2 на дефиниция 7 е изпълнено, когато и двата върха v, w принадлежат на един и същи граф D_{ij}, $j = 1, \ldots q$. Нека $(v, w) \in E(G_i)$ и $v \in V(D_{ip})$ $w \in V(D_{is})$ $v = [x, v_j]$ $w = [x_1, w_j]$. Следователно $(x, x_1) \in E(G_{i-1})$, а v_j е връх съседен на изходно терминално ребро в граф D_{ip} от цвят $col(x, x_1)$, а w_j е връх съседен на входно терминално ребро в D_{is} от цвят $col(x, x_1)$. Тогава $S_i(v) = (S_{i-1}(x), S_0(v_j))$

$S_i(w) = (S_{i-1}(x_1), S_0(w_j))$, но $(S_{i-1}(x), S_{i-1}(x_1)) \in E(H_{i-1})$, следователно $(S_i(v), S_i(w)) \in E(H_i)$. От (4.4) $T_i(v) = T_i(w) - 1$, следователно $(M_i(v), M_i(w)) \in E(G_i')$. С това условие 2 на дефиниция 7 е доказано. Остава да се докаже условие 4 на дефиниция 7. Т.е. ако v е терминален връх за G_i, то $pr(M_i(v))$ е терминален връх за H_i. От това, че v е терминален връх за G_i следва, че v е терминален връх за някой от хомогенните графи $D_{i,j}$, $j = 1, \ldots q$, получени от G_i. От аналогичната теорема за хомогенни графи следва, че $pr(M_i(v))$ е терминален връх за някой от хомогенните графи получени от H_i. Ако допуснем, че образа на връх v не е терминален за H_i, то се нарушава условие 1 на дефиниция 7. Следователно изображението M_i е систолическа реализация. □

4.3 Заключение

В тази глава разгледахме конструирането на систолически масиви използвайки техника за разширяване. Първо разгледахме хомогенни систолически масиви (систолически масиви състоящи се от еднотипни клетки). Тези масиви отговарят на задачи в които се извършват един тип изчисления. След това са разгледани хетерогенни систолически масиви (систолически масиви състоящи се от различни видове клетки). Тези масиви отговарят на задачи в които се извършват различни видове изчисления. Намерили сме необходимо и достатъчно условие един хомогенен систолически масив да е разширим, а след това и един хетерогенен систолически масив да бъде разширим. Доказали сме коректността на алгоритъма за разширяване.

Библиография

[1] Systolic arrays eds: Moore W., McCabe A., Urquhart R., Adam Hilger, Bristol ,1982.

[2] **Akl S.G., Calvert M., Stojmenovic I.,** Systolic generation of derangements, *Algorithms and parallel VLSI architectures II, Elsevier Sci. Pub.* , 1992, pp.59-70.

[3] **Andonov R.A., Benaini A.,** A linear Systolic Array for the 0/1 Knapsack Problem, *TR 90-12, LIP-IMAG, Ecol Normale Superieur de Lyon*, 1990.

[4] **Andonov R., Gruau A.,** A 2D toroidal systolic array for knapsack problem, *In Algorithms and Parallel VLSI Architectures II, Bonas, France*, June 1991.

[5] **Andonov R., Quinton P.,Rajopadhye S.,Wilde D.** A Shift-Register based systolic array for the general knapsack problem , *Parallel Procesing Letters, 5(2)*, 1995.

[6] **Andonov R., Rajopadhye S.,** Knapsack on VLSI:From Algorithm to Optimal Circuit, IEEE Transaction on Parallel and Distributed Systems, vol 8(6),1997

[7] **Aleksandrov V., Fidanova S.,** On the average execution time for a special class of non uniform recurrence equation on linear systolic array, *Int. conf. of Num. meth. and applications, Sofia*,1992.pp.144-152.

[8] **Aleksandrov V., Fidanova S.,** On the expected exeqution time for a class of non uniform recurrence equation mapped onto 1D regilar array,*J. of Parallel Algorithms and Applications, Vol. 1.*,1994, pp303-314.

[9] **Aleksandrov V., Fidanova S.,** Optimal 2D regular array for a special class of non uniform recurrence equations, *Tech. report, N 634, University of Newcastle, UK*,1993,17pp.

[10] **Aleksandrov V., Fidanova S.,**Non uniform Recurrence equations on 2D regular arrays, *Advances in numerical methods and applications,EDs Iv. Dimov, Bl.Sendov, P. Vasilevski,World Scientific,*1994.

[11] **Bollobas B.**,Graph Theory,*Springer-Verlag,New York,*1979.

[12] **Chen M.C.,** Synthezing Systolic Designs,*Intern. Symp. on VLSI Techn., Taipei, Taiwan,* 8-10 May, 1985.

[13] **Chen G.H., Chern M.S., Jang J.H.,** Pipeline architectures for dynamic programming algorithms, *Parallel Computing 13*, 1990, pp.111-117.

[14] **Chow E., Hunkapiller T., Peterson J.,** Biological Information Sygnal Processor, *In ASAP*, 1991, pp.144-160.

[15] **Cormen T.H., Leiserson C.E., Rivest R.L.,** Introduction to Algorithms, *The MIT Press*, 1990

[16] **Chen M.C.,** The generation of a class of multiplier: synthetizing highly parallel algorithms in VLSI,*IEEE Trans. Computing, Vol C-37*, 1988,pp.329-338.

[17] **Delosme J.-M., Ipsen I.C.F.** Systolic arrays, *Adam Hilger, Bristo*,1987.

[18] **Djidjev H. N.,** Highly Parrallel Computers, *North-Holand,* 1987, pp.157-174.

[19] **Djidjev, H. N.**, Design and Analysis of Systolic Implementations, *J. of New Gener. Comput. Syst. 1*, 1988, pp.7-20

[20] **Fidanova S.**, The mapping of algorithms on linear systolic array,*Int. konf. of Parallel and distributed processing,Sofia*,1991

[21] **Fidanova S.**, Recursive algorithms on fixed size systolic array, *Int. conf. of Parallel and distributed processing, Sofia*, 1993.pp 360-368

[22] **Fidanova S.**,Automatic derivation of expandable systolic algorithms for LU decompozition, *Int. conf. of Mathematical Modeling and Scientific Computation, Sozopol*, 1993.

[23] **Fidanova S.,Aleksandrov V.,Megson G.**,Mapping Knapsack Type Problems on 2D Regular Arrays: Two Case Studies, *Int. konf. of Parallel Processing, Potsdam, Germany*, 1994,17p.

[24] **S.Fidanova**,Linear array for spelling correction,*Concurency:Practise and Experience, Vol 7*,1997

[25] **Fischer A.L., Kung H.L.**, Synchronizing large VLSI processor array, *Tenth Annual Ieee/ACM Symposium on computer architectures*, 1983, pp.54-58.

[26] **Flynn M.J.**, Some computer organisation and their effectiveness, *IEEE Trans Computers 21,9*, 1972, pp.948-960.

[27] **Fortes J.A.B., Moldovan D.I.**, Parallelism detection and transformation techniques useful for VLSI algorithms, *J. Parallel and Distributed Computing*, 1985.

[28] **Garey M., Jonson D.**, Computers and Intractability: A Guide to the Theory of NP-Completeness, *Freeman, San Francisco*, 1979

[29] **Garfinkel R., Nemhauser G.,** Integer Programming, *Wiley, New York.* 1972.

[30] **Kung H.T.,** Why systolic architectures?, *Computer 15(1),* 1982, pp.37-46.

[31] **Kung H.T.,** The structure of parallel algorithms, *Adv. Computer 19,* 1980, pp.65-112.

[32] **Kung H.T., Leiserson C.E.,** Systolic array for VLSI, *In Introduction to VLSI Systems, C. Mead and L. Conway, Addison-Wesley reading mass.,* 1980, pp.260-292.

[33] **Kung H.T., Lin W.T.,** Elliptic problems slovers. *Academic Press,* 1984, pp.141-160.

[34] **Kung H.T., Lo S.C., Lewis P.S.,** Optimal systolic design for the transitive closure and the shortest path problems, *IEEE trans Comput. C-36(5),* 1987, pp.603-614.

[35] **Kung S.Y.,** VLSY Array processors, *Prentice Hall, New Jersey,* 1988.pp.198-282.

[36] **Karp R. M., Miller R. E., Winograd S.,** The Organization of Computation for Uniform Recuence Equation, *JASM 14, Juil,* 1967,pp.563-590.

[37] **Lamport L.,** The Parallel Exeqution of DO-loops, *Comp. ACM. Vol 17 No2,* Feb. 1974, pp.83-93.

[38] **Lavenier D.,** Dedicated Hardware for Biological Sequence Comparison, *J. of Universal Computer Science, Vol 2(2),* 1996, pp.77-86.

[39] **Lee J., Shragowitz E., Sahni S.,** A hypercube algorithm for the 1/0 knapsack problem, *J. of Parallel and Distributed Computing 5,* 1988, pp.483-456.

[40] **Lee P., Kedem Z.,** Synthezing linear array algorithms from nested for loop algorithms,*IEEE Trans Comput., C-37,* 1988, pp.1578-1598.

[41] **Li G.H., Wan B.W.,** The design of optimal systolic arrays, *IEEE Trans. Comp.34(1)*, 1985, pp.66-77.

[42] **Lin J., Storer J.,** Processor efficient hypercube algorithm for the knapsack problem, *J of Parallel and Distributed Computing 13*, 1988, pp.332-337.

[43] **Lisper B.,** Computing tranzitive closure on systolic arrays of fixed size, *Distributed Computing 5*, 1991, pp.133-144.

[44] **Martello S., Toth P.,** Knapsack Problems: Algorithms and Computer Implementation, *Jon Wiley,* 1990

[45] **Moldovan D. I., Fortes J.A.B.,** Partitioning and Mapping of Algorithms into Fixed Size Systolic Arrays, *IEEEE Trans. on Computers, 35(1)*, 1986, pp.1-12.

[46] **Moldovan D. I.,** On the Analysis and Synthesis of VLSI Algorithms, *IEEE Trans. Comput. C-32*, 1982, pp.1121-1126.

[47] **Mongenet C., Clauss Ph., Perrin G.-R.,** Geometrical Tools to Map Systems of Affine Recurrence Equations on Regular Arrays, *Parallel Architectures and Languages Europe, PARLE91, LNC505, Springer Verlag Ed.*, 1995

[48] **Myoupo J.-F., Fabret A.-C.,** Designing modular linear systolic array using dependance graph partition, *IEEE Transaction on Parallel and Distr. Systems*, 1995.

[49] **Quinton P.,** Automatic synthetisis of systolic arrays from uniform recurrent equation, *IEEE 11th Symp. on Computer Architecture. ANN*, 1984, pp.208-214.

[50] **Quinton P.,** Mapping recurrences on parallel architectures, *Third Intern. Conf. Supercomputing, Boston*, 1988, pp.15-20.

[51] **Quinton P., Van Dongen V.,** The Mapping of Linear Recurence Equation on Regular Arrays, *Journal of VLSI Signal Processing*, 1989.

[52] **Quinton P., Van Dongen V.,** The mapping of linear recurrence equations of regular arrays, *J. of VLSI Signal Processing 1*, 1989, pp.95-113.

[53] **Radjopadhye S., Fudjimoto R.,** Synthesing Systolic Arrays for Recurrence Equations, *Parallel Computing 14, North-Holand,* 1990,pp.163-189.

[54] **Ramakrishnan I.V., Varman P.J.,**On mapping cube graphs onto VLSI arrays, *Lectures Notes in Computer Science, Springer-Verlag, Berlin-Heidelberg-New York-Tokio*, 1984,pp. 296-316.

[55] **Ramakrishnan I.V., Fussel D. S., Silberschatz A.,** Mapping homogeneous graphs on linear arrays *IEEE Transactions on Computers C-35*,1986, pp.189-209

[56] **Stoimenovic I.,** An optimal algorithm for generating equivalence relations on linear array of processors, *BIT 30(3),* 1990,pp.424-436.

[57] **Schwiegelshohn U., Thiele L.,** Linear systolic array for matrix computations, *J. Parallel and Distributed Coputing Vol.7*, 1992,pp.28-39.

[58] **Teng S.,** Adaptive Parallel Algorithm for Integral Knapsack Problem, *J of Parallel and Distributed Computing, 8*,1990, pp.400-406.