ПРОГРАММА МОДЕЛИРОВАНИЯ АЛГОРИТМА ПАРАЛЛЕЛЬНЫХ ПОДСТАНОВОК И ПРИМЕР ЕГО РЕАЛИЗАЦИИ НА ПЛИС

Баранов Е.Ю.
РГАТА имени П.А.Соловьева
john851@yandex.ru
Мирзоян А.С.
РГАТА имени П.А.Соловьева
amspectre@gmail.com
Петров А.В.
РГАТА имени П.А.Соловьева
gmdidro@gmail.com

Аннотация

Приводится описание основных понятий алгоритма параллельных подстановок, иллюстрация его применения на примере сложения множества положительных двоичных чисел. Описывается структура программы моделирования исполнения алгоритма параллельных подстановок, основанная на технологии клиент-сервер. Приводится описание языка программирования QuickPSA и его транслятора на язык Python. Раскрывается реализация на ПЛИС алгоритма параллельных подстановок на примере сложения множества положительных двоичных чисел.

1. Введение

Основой реализации большинства последовательных вычислительный устройств считается конечный автомат. Сегодня увеличение пиковой производительности вычислительных устройств уже не возможно за счет увеличения тактовой частоты их работы, поскольку достигнут технологический предел её роста [1]. В качестве основной линии развития современных вычислительных систем можно выделить общее изменение структуры вычислительного устройства с акцентом на увеличение возможностей параллельного исполнения программ. В качестве примеров, подтверждающих эту тенденцию, можно привести многоядерные процессоры, GPU-вычислители, вычислители на основе архитектуры CELL BE. В тоже время развиваются и специализированные вычислители, основанные, например, на использовании ПЛИС в качестве сопроцессоров [2]. Среди множества подходов к построению параллельных вычислительных систем можно выделить концепции крупноблочного и мелкозернистого параллелизма. Идея мелкозернистого параллелизма была реализована в частности в вычислительных устройствах систолической архитектуры, например, в транспьютерах iWarp фирмы Intel и SeaForth фирмы Intellasys.

К другой разновидности мелкозернистого параллелизма можно отнести идею алгоритма параллельных подстановок, которая была предложена группой О.Л.Бандман [3]. Нами был создан язык QuickPSA написания программ, предназначенных для исполнения по принципу алгоритма параллельных подстановок, интерпретатор языка QuickPSA, моделирующий параллельное исполнение программ на сервере и представляющий результаты моделирования на клиентских компьютерах и пример реализации одной из таких программ на ПЛИС.

2. Описание алгоритма параллельных подстановок (АПП)

Пусть $\boldsymbol{A}$ – конечный алфавит, а $\boldsymbol{M}$ – множество имен (с мощностью не более чем счетной). ***Клетка*** - пара $\boldsymbol{(a, m)}$, принадлежащая множеству $A \times M$. ***Клеточный массив W (слово)*** – конечная совокупность клеток, в которой нет ни одной пары клеток с одинаковыми именами. Векторы

$\Pr 1(W) = (a_0 a_1 ... a_n)$ и $\Pr 2(W) = (m_0 m_1 ... m_n)$ называются ***первой и второй проекциями слова*** W соответственно.

Конфигурация задается в виде: $S = \{(a_1, \varphi_1(m))...(a_n, \varphi_n(m))\}$, где $a_i \in A$, $m \in M$, где $\{\varphi_1(m),...,\varphi_n(m)\}$ – упорядоченная совокупность функций, отображающих $M \to M$, причем $\forall m \in M$ выполняется условие $\varphi_i(m) \neq \varphi_j(m)$, при $i \neq j$, $i, j = 1,\ldots,n$. Конфигурацию можно представить в наглядной геометрической форме, если множество имен M интерпретировать точками какого либо пространства, что будет продемонстрировано в примере ниже.

Подстановка задается в виде $\Pi : S_1 * S_2 \to S_3$, где S_1, S_2, S_3 – конфигурации, S_1 – левая часть, S_2 – контекст, S_3 – правая часть.

П применима к ***W,*** если ***W*** содержит ее левую часть, т. е. существует такое $x_k \in M$, что $S_1 * S_2(x_k) \in W$. ***Результат применения подстановки*** Π к ***W:*** $\Pi(W) = [W \setminus (W \cap S_1)] \cup S_3$.

Система параллельных подстановок (СПП) – конечное множество подстановок, записанных в производном порядке $\Phi = \{\Pi_1,\ldots,\Pi_n\}$. Система задает преобразование слова W следующей итерационной процедурой:

1. Если ни одна подстановка $\Pi_i \in \Phi$ не применима к $W^{(i-1)}$, то $\Phi(W) = W^{(i=1)}$ является результатом применения Φ к W;

2. Если существуют подстановки $\Pi_i \in \Phi$ применимые к $W^{(i-1)}$, то

$$W^{(i)} = \left(W^{i-1} \setminus \bigcup_{j=1}^{n} S_{j1} \right) \bigcup \left(\bigcup_{j=1}^{n} S_{j3} \right) \qquad (1)$$

АПП (Алгоритм Параллельных Подстановок) – система параллельных подстановок с вышеприведенной процедурой их применения.

Основные идеи АПП:

- Обрабатываемая информация задается в виде клеточного массива;
- Каждая клетка – данные (бит, символ и т. д.) с определенными координатами в массиве;
- Алгоритм задается системой подстановок, имеющих левые и правые части;
- Если СПП непротиворечива, то выполнять подстановки можно в любом порядке, последовательно или параллельно.

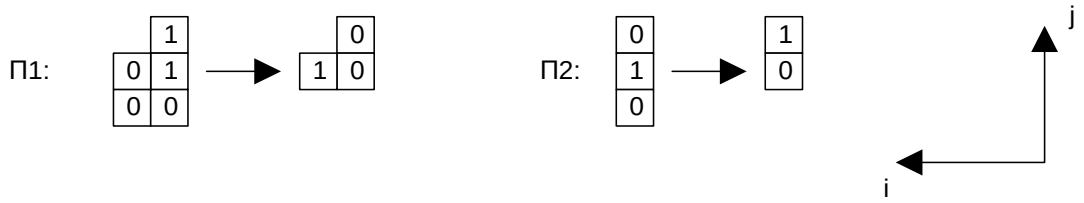
3. Пример СПП для сложения массива положительных чисел

Рассмотрим пример, иллюстрирующий описание системы параллельных подстановок для сложения массива положительных двоичных чисел [3].

Исходные данные: A={0, 1}, $M=N\times N$, W – это двумерная прямоугольная таблица, клетки которой пронумерованы в соответствии с левой системой координат (i – абсцисса, j – ордината). В строках таблицы расположены двоичные слагаемые. Размеры таблицы определяются числом слагаемых и их разрядностью. Нижняя строка таблицы нулевая. Младшие разряды слагаемых записаны в столбце с i=1. Система подстановок имеет вид:

$$\Phi = \begin{cases} \Pi_1 : \{(1,\langle i, j\rangle)(1,\langle i, j+1\rangle)(0,\langle i+1, j\rangle)\} * \{(0,\langle i, j-1\rangle)(0,\langle i+1, j-1\rangle)\} \to \{(0,\langle i, j\rangle)(0,\langle i, j+1\rangle)(1,\langle i+1, j\rangle)\}; \\ \Pi_2 : \{(1,\langle i, j\rangle)(0,\langle i, j+1\rangle)\} * \{(0,\langle i, j-1\rangle)\} \to \{(0,\langle i, j\rangle)(1,\langle i, j+1\rangle)\}. \end{cases}$$

Геометрические образы конфигураций обоих подстановок имеют вид:



Ниже будут представлены варианты реализации данного примера на языке QuickPSA для моделирования на ПК и на языке Verilog для исполнения на ПЛИС.

4. Описание языка QuickPSA

Для разработки и отладки исполнения систем параллельных подстановок, реализующих тот или иной вычислительный алгоритм удобно использовать программы, моделирующие параллельное исполнение СПП на ПК. Примером такой программы может служить система WinALT [4]. Нами было принято решение о разработки собственной программы моделирования исполнения АПП, поскольку исходные коды WinALT не предоставляются в открытом доступе, что в свою очередь затрудняет модификацию данной системы. Описание программы, разработанной нами, приводится ниже в разделе 5.

Входные данные для программы задаются на языке Python, что может быть не вполне удобно для пользователя, работающего с формализмами АПП. Язык QuickPSA был разработан для более удобного способа описания СПП. Этот язык ориентирован на полное описание задачи в терминах АПП (алфавиты символов, входное слово, функции для переменных символов, описание системы параллельных подстановок). QuickPSA транслируется в код на языке Python, после чего исполняется моделирующей программой. В качестве инструмента написания транслятора языка QuickPSA был использован инструмент ANTLR [5] (Another Tool for Language Recognition), который позволяет с помощью специального вида записи РБНФ задавать грамматику разрабатываемого языка, и реализовывать кодогенерацию в терминах инструкций обхода абстрактного синтаксического дерева, что дает существенное упрощение разработки транслятора.

Пример описания СПП для сложения массива положительных чисел на языке QuickPSA приведен ниже.

```
psa Sum_prog
{// входное слово
inputword: 0 in [0,0] | 0 in [1,0] | 0 in [2,0] |
           1 in [0,1] | 0 in [1,1] | 0 in [2,1] |
           0 in [0,2] | 1 in [1,2] | 0 in [2,2] ;
sub sub1 // первая подстановка
{
  S1: 1 in [x,y]     | 1 in [x,y+1] | 0 in [x+1,y];
  S2: 0 in [x+1,y-1] | 0 in [x,y-1];
  S3: 0 in [x,y]     | 0 in [x,y+1] | 1 in [x+1,y];
}
sub sub2 // вторая подстановка
{
  S1: 1 in [x,y] | 0 in [x,y+1];
  S2: 0 in [x,y-1];
  S3: 0 in [x,y] | 1 in [x,y+1];
}
}
```

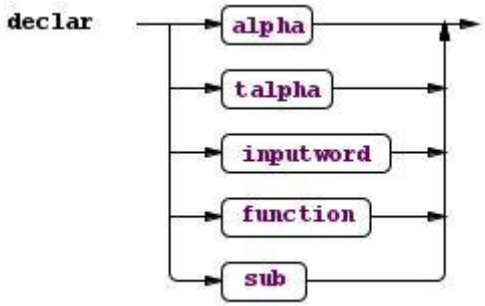
Краткое описание основных конструкций языка QuickPSA с помощью синтаксических диаграмм приведено ниже.

- Правило, задающее форму описания имени и тела алгоритма параллельных подстановок:
  – Синтаксическая диаграмма

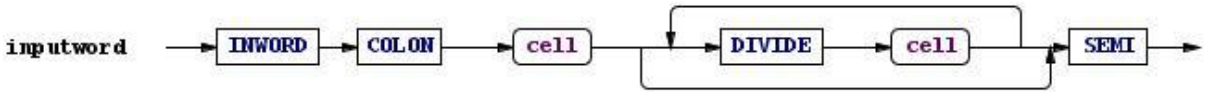
  – Пример на языке QuickPSA

```
psa <имя_СПП> {<тело СПП>}
```

- Правило, задающее форму описания тела СПП:
  – Синтаксическая диаграмма



- Правило, задающее форму описания входного слова:
  – Синтаксическая диаграмма



  – Пример на языке QuickPSA

```
inputword: a in [1,3] | b in [6,8] | c in [7,7];
```

- Правило, задающее форму описания одной подстановки:
  – Синтаксическая диаграмма

  – Пример на языке QuickPSA

```
sub sub1
{
  S1: null;
  S2: a in [x+1,y,z-1] | 1 in [x,y+10,z*10];
  S3: 3 in [x,y,z] | b in [x+1,y+2,z+3];
}
```

5. Программа моделирования исполнения АПП

Для разработки и отладки исполнения систем параллельных подстановок, реализующих тот или иной вычислительный алгоритм, была разработана программа, моделирующая параллельное исполнение программ на сервере и представляющий результаты моделирования на клиентских компьютерах. Структура программы изображена на рисунке

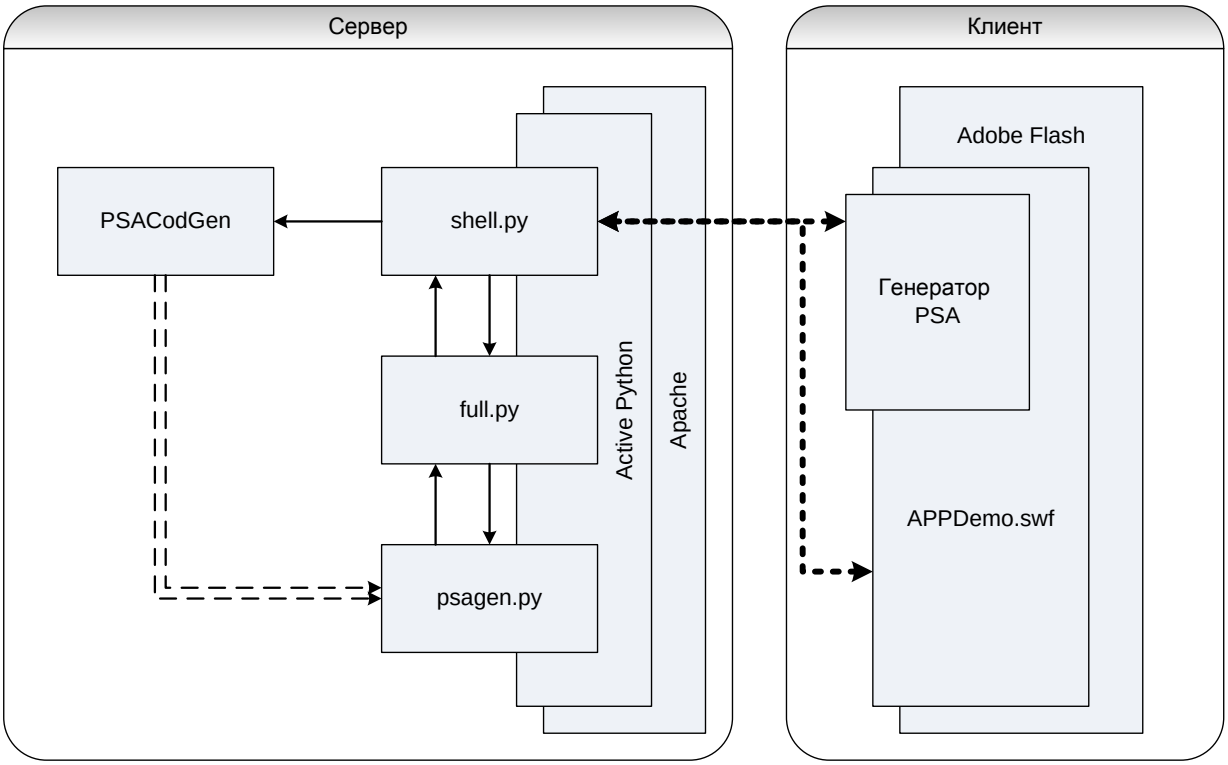


Рис. 1. – Структура клиент-серверной программы моделирования исполнения СПП.

Взаимодействие строится на основе клиент-серверного взаимодействия с пассивным сервером. В общем случае клиент и сервер могут находиться на различных компьютерах. Клиент является лишь компонентой представления. Цель клиента – наглядное представление исходных данных и результата алгоритма параллельных подстановок. Задача сервера – непосредственное выполнение алгоритма на языке параллельных подстановок QuickPSA. Общение клиента и сервера происходит посредством протокола TCP/IP на стандартном HTTP-порте.

Клиент использует мультимедийную платформу Adobe Flex [6], разработан в среде Adobe Flex Builder на языке MXML. Язык MXML –декларативный язык описания интерфейсов, который используется платформой Adobe Flex. Adobe Flex, как правило, применяется для создания Интернет-приложений со сложным пользовательским интерфейсом (Rich Internet Applications, RIA). MXML используется для настройки платформы, описания и настройки свойств элементов интерфейса, например кнопок, таблиц и т.п. В MXML можно включать каскадные таблицы стилей (CSS) и скрипты ActionScript 3, что очень удобно для обработки событий. По сути MXML является декларативной надстройкой над ActionScript, так как при компиляции по MXML-файлам создаются эквивалентные ActionScript-файлы, которые в дальнейшем уже переводятся в бинарный код.

На рисунке ниже изображен клиент, отображающий результаты моделирования исполнения алгоритм сложения двоичных чисел.

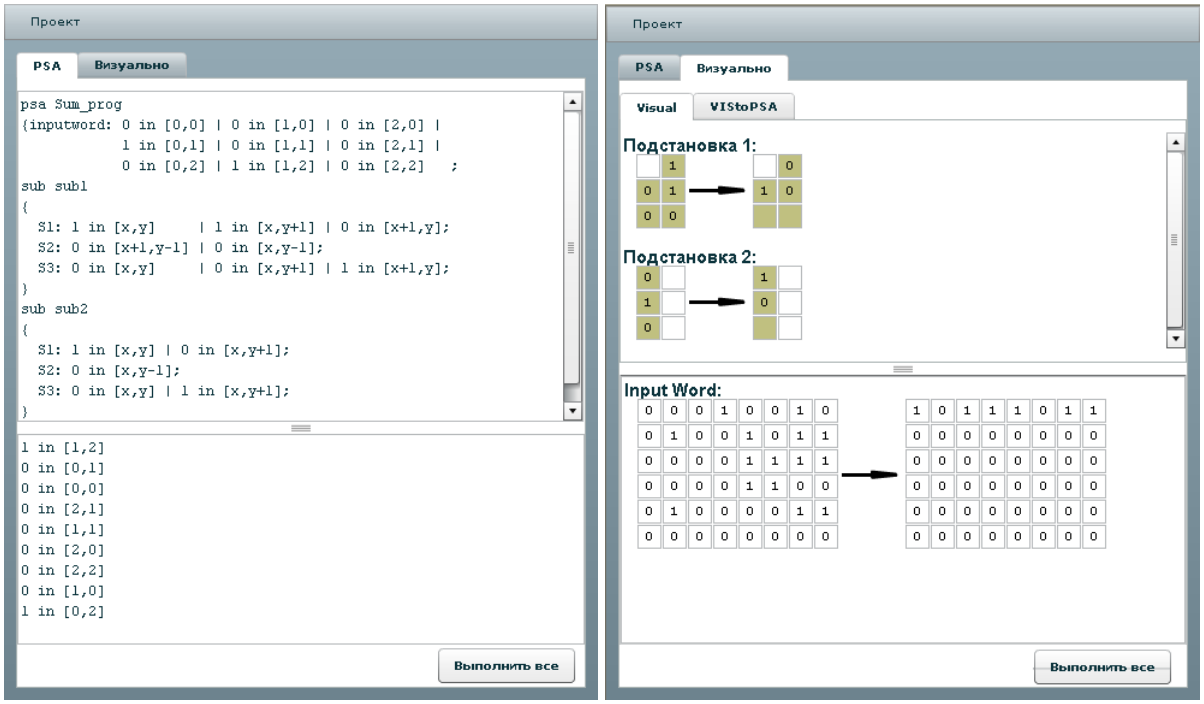


Рис. 2. - Внешний вид клиента. а) описание на QuickPSA; б) описание в наглядном виде.

Пользователь может описывать СПП либо прямо в языке QuickPSA (Рис. 2, А), либо в наглядном представлении (Рис. 2, Б). Во втором случае перед отправлением данных на расчет на сервер, происходит генерация QuickPSA.

На сервере установлен Web-сервер Apache с ActivePython. ActivePython - расширенный дистрибутив интерпретатора языка Python. Python – гибкий высокоуровневый язык программирования, поддерживающий множество парадигм программирования, динамическую типизацию, сборку мусора и многопоточные вычисления. Также Python включает обширную библиотеку подпрограмм вплоть до анализа функций и построения моделей. Сервер выполняет моделирование исполнения СПП и в качестве ответа возвращает результат алгоритма в XML-представлении.

На сервере находятся 4 скрипта:

1) shell.py – головной скрипт, вызываемый клиентом. Назначение – организация всего процесса, запуск скриптов PSACodGen и full.py;
2) PSACodGen – Java-программа, транслирующая описание системы параллельных подстановок с языка QuickPSA на Python (psagen.py). Более подробное описание см. в разделе 4;
3) full.py – скрипт-оболочка, непосредственно выполняющий алгоритм, используя полученный psagen.py. В данном скрипте происходит также анализ алгоритма на применимость во время исполнения и генерация результата в XML-представлении;
4) psagen.py – сгенерированный скрипт, содержащий описание входных данных и функций, задающих подстановки.

В основе алгоритма моделирования исполнения СПП лежит формула (1) (см. раздел 2). Алгоритм реализован на языке Python. Благодаря функциональным возможностям данного языка реализация получилась достаточно компактной. Исходный код функции, реализующей шаг моделирования приведен ниже:

```
def makeOneStep(self,word,system):  # функция, применяющая СПП system к входному слову word
    S1resulted=[] # объединение правых частей конкретизированных
                  # подстановок СПП (за вычетом контекста)
```

```
S3resulted=[] # объединение левых частей конкретизированных подстановок СПП
for rule in system : # для каждой подстановки из СПП
    wordspivots=self.getPivots(word,rule) # найти клетки во входном слове word, относительно
                                          # которых возможна конкретизация подстановки rule
    for pivot in wordspivots:   # для каждой «опорной» клетки во входном слове
        concr=rule(pivot ,word) # конкретизировать подстановку
        S1resulted=S1resulted + concr['s1'].items()
        S3resulted=S3resulted + concr['s3'].items()

set1=set(word.items()) # выполнить действия над множествами по формуле (1)
set2=set(S1resulted)
set3=set(S3resulted)
resSet=(set1-set2)|set3
return dict(resSet)
```

6. Реализация АПП сложения двоичных положительных чисел на ПЛИС

Для демонстрации возможности аппаратной реализации примера сложения двоичных положительных чисел на практике использовалась ПЛИС (Программируемая Логическая Интегральная Схема) – особый вид программируемых микросхем, которые меняют свою внутреннюю структуру в зависимости от прошивки, что позволяет быстро и легко реализовывать цифровые электрические схемы разной степени сложности.

Пример был реализован с использованием языка Verilog HDL. Он относится к классу языков описания аппаратуры, которые разработаны для проектирования цифровых электронных схем. Основное отличие данного класса от других алгоритмических языков состоит в том, что код программы не компилируется в команды для процессора. После компиляции формируется специальный файл, в котором формально описана электронная схема. Данное описание используется для конфигурирования ПЛИС.

Для компиляции описания на языке Verilog используют различные САПР (Система Автоматического ПРоектирования). САПР - программный пакет, осуществляющий весь цикл проектирования схемы на разных уровнях абстракции: поведенческий, уровень регистровых передач, уровень логических вентилей. При реализации данного примера использовалась САПР Quartus II от фирмы Altera.

Для реализации АПП сложения двоичных положительных чисел было решено использовать клеточный автомат [7]. Каждая клетка в клеточном массиве – конечный автомат, с внутренним состоянием (1 или 0), множеством входов/выходов и функцией перехода.

Логику работы каждой клетки можно описать на основе принципа “центра” подстановки (рисунок 3).

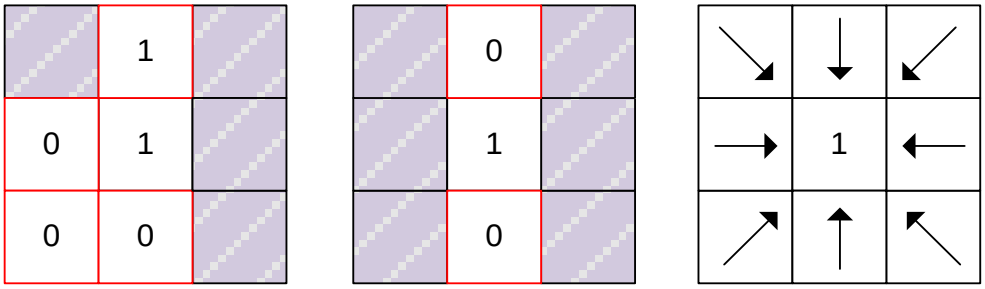


Рис. 3. – “Центр” подстановки.

Клетка анализирует свое состояние, если оно является логической единицей, то клетка потенциально может стать “центром” подстановки. После чего проверяется состояние соседей, расположение которых определено конфигурациями подстановки, затем клетка либо определяет, что возможна подстановка и пересылает управляющие сигналы соседним клеткам, либо меняет свое состояние под воздействием соседней клетки, либо бездействует, сохраняя прежнее состояние.

Исходный код аппаратного модуля, реализующего логику работы одной клетки, приведен ниже на языке Verilog:

```
module one_cell
#(parameter in_wid=5,out_wid=5)
(
 output reg [out_wid:0] out,              //сигнал соседям о текущем состоянии
                                            клетки
 output reg             interface_out,    //вывод состояния клетки на индикацию
 output reg [out_wid:0] force_init_out,   //выход для принудительного задания
                                            состояния соседним клеткам
 output reg [out_wid:0] init_state_out,   //значение принудительного состояния

 input      [in_wid:0]  force_init_in,    //сигнал о необходимости принудительной
                                           записи состояния
 input      [in_wid:0]  init_state_in,    //состояние, которое требуется записать

 input      [in_wid:0]  in,               //входной сигнал о состоянии 4х соседних
                                            клеток
 input                  interface_state, //вход, используемый для начальной
                                            инициализации
 input                  enable_sub,      //входы для управления краевыми случаями
                                           (границы клеточного массива)
 input                  enable_2sub,
 input                  reset,           //общий сброс
 input                  clk              //общий тактирующий вход
);
reg cur_state, next_state;                //регистр, хранящий текущее состояние
//комбинационная схема формирования следующего состояния
always
@(force_init_in,in,init_state_in,interface_state,cur_state,enable_sub,enable_2sub)
begin
  //инициализация выходов
  out={6{cur_state}};
  interface_out=cur_state;
  force_init_out=0;
  init_state_out=0;
  next_state=cur_state;
  //проверка: не пришел ли сигнал от соседних клеток сменить состояние?
  if (force_init_in[0]) next_state=init_state_in[0];
  else if (force_init_in[1]) next_state=init_state_in[1];
  else if (force_init_in[2]) next_state=init_state_in[2];
  else if (force_init_in[3]) next_state=init_state_in[3];
  else if (force_init_in[4]) next_state=init_state_in[4];
  else if (force_init_in[5]) next_state=init_state_in[5];
  //если не пришел то, проверяем, является ли данная клетка "центром" подстановки
  else if (cur_state && enable_sub)
  begin
  //проверка соседних клеток на применимость подстановки 1
  if (~in[0] && ~in[3])
            begin
              next_state=0;

              force_init_out[0]=1;
```

```
            init_state_out[0]=1;
          end
    //проверка соседних клеток на применимость подстановки 2
    else if (in[0] && ~in[3] && ~in[4] && ~in[5] && enable_2sub)
          begin
            next_state=0;

            force_init_out[0]=1;
            init_state_out[0]=0;

            force_init_out[5]=1;
            init_state_out[5]=1;
          end
end
end

//последовательная схема смены состояний
always @(posedge clk, negedge reset)
begin
 if (~reset) begin cur_state<=interface_state; end
 else cur_state<=next_state;
end
endmodule
```

При реализации примера использовался массив из девяти клеток, объединенных между собой следующим образом - рисунок 4.

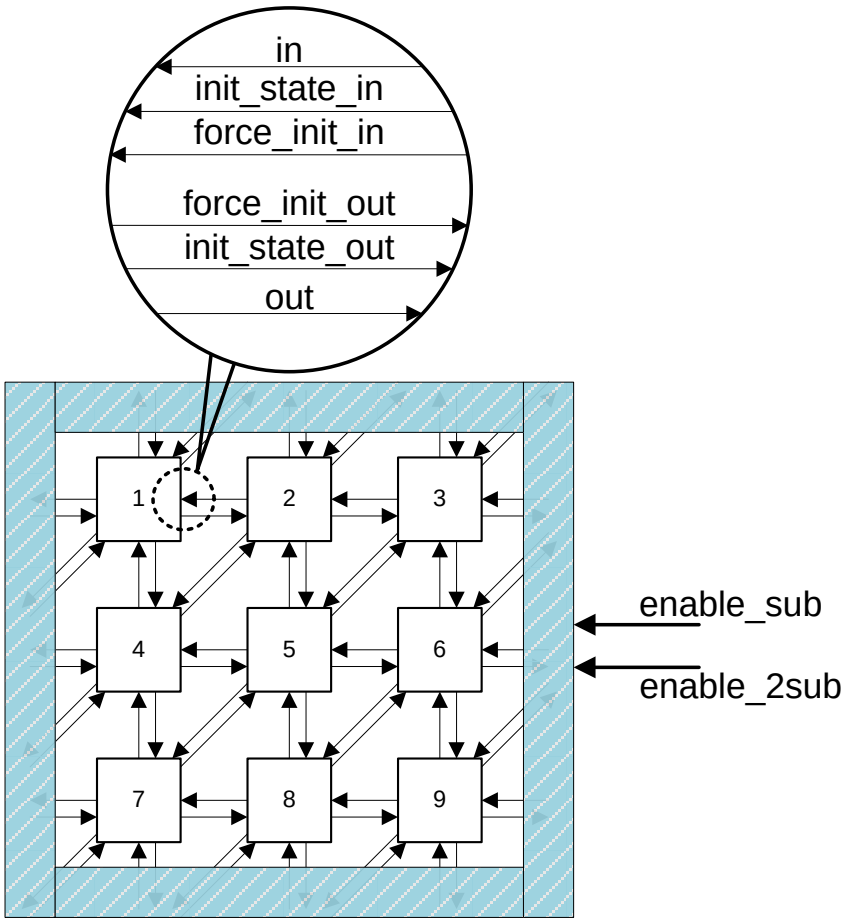


Рис. 4. – Способ объединения клеток между собой.

Следует отметить, что когда клетки работают в массиве, каждая из них выполняет свои функции параллельно с другими клетками во всем массиве, тем самым достигается параллелизм вычислений на каждом шаге выполнения АПП для сложения множества положительных чисел.

Данный пример позволяет продемонстрировать возможность создания специализированных вычислительных устройств на основе принципов алгоритма параллельных подстановок.

Для тестирования примера использовались следующие средства:

1. отладочная плата Altera DE2 Board на базе ПЛИС Cyclone II (EP2C35);
2. САПР Quartus II;
3. симулятор ModelSim.

7. Заключение

Идея совмещения алгоритма подстановок Маркова и клеточных автоматов, реализованная в алгоритме параллельных подстановок, представляется чрезвычайно интересной. В данной статье были продемонстрированы некоторые результаты, позволяющие сделать предположение о возможности создания технологии конструирования специализированных вычислителей на основе принципов алгоритма параллельных подстановок. Однако на данном этапе еще множество вопросов остаются не решенными, и требуется дополнительное время для их исследования.

Литература

1. So long Moore; Hello multi-core // http://h41112.www4.hp.com/promo/blades-community/eur/en/library/articles/So_long_moore.pdf
2. Суперкомпьютеры на основе ПЛИС // http://fpga.parallel.ru/supercomputers.html
3. Анишев П.А., Ачасов М.С, Бандман О.Л., Пискунов С.В., Сергеев С.Н., Методы параллельного микропрограммирования / Под ред. О.Л. Бандман.- Новосибирск: Наука, Сибирское отделение, 1981
4. Официальный сайт WinALT // http://winalt.sscc.ru
5. Официальный сайт ANTLR // www.antlr.org
6. Официальный сайт Adobe Flex //www.adobe.com/ru/products/flex/
7. Delorme M., Mazoyer J., Cellular Automata, A Parallel Model. Cluwer Academic Publishers, 1999.