ИПМ им. М. В. Келдыша РАН

Технологии разработки прикладного ПО для реконфигурируемых вычислительных структур (ПЛИС).

Андреев С. С., Давыдов А. А., Дбар С. А., Лацис А. О., Плоткина Е. А.

lacis@kiam.ru

2010

Общий план доклада:

- Что такое ПЛИС.
- Место нетрадиционных вычислителей вообще и ПЛИС в частности в сегодняшней HPC – проблематике.
- Что предлагается сегодня.
- Что мы предлагаем сегодня.
- Что мы предлагаем завтра.

Что такое ПЛИС.

- **И**нтегральная **С**хема **П**рограммируемой **Л**огики.
- Любая цифровая схема строится из элементов некоторого базового набора.
- Все базовые наборы эквивалентны и универсальны.
- Если на пластине кремния изготавливаются необходимые для данной схемы базовые элементы и связи между ними, то это – ИС жесткой (не программируемой) логики.
- Если же изготавливается «впрок» много базовых элементов и система коммутации, которая позволяет потом соединить их так, как захочет пользователь, то это – ИС программируемой логики, или ПЛИС.
- Наиболее мощные ПЛИС, со многими миллионами базовых элементов и сотнями готовых типовых логических узлов, называются **FPGA** (**F**ield **P**rogrammable **G**ate **A**rray).
- Время конфигурирования, или «прожига» - десятки секунд, прожигать можно сколько угодно раз.

Области применения.

- Как современный вариант «макетной платы» при разработке новых микросхем.
- Как деталь штучных и мелкосерийных (тираж << 1 млн. штук) изделий.
- Как реконфигурируемый «процессор одной задачи» для схемной реализации вычислений (наш случай).

Но почему именно сейчас?

- Успешные эксперименты по прямой схемной реализации вычислений делались и 20 лет назад. Но оставались экзотикой.
- Что произошло примерно 3 года назад, когда внезапно и резко обострился интерес к вычислителям нетрадиционной архитектуры?
- «Виновато» не желание жить по-новому, а невозможность жить по-старому.
- Число базовых элементов на кремниевой пластине окончательно «переросло» возможности фоннеймановской архитектуры.
- Во весь рост встала проблема доминирования вспомогательных операций во времени выполнения программ вычислительного характера, также известная как «проблема паровозного к.п.д.». «Стена памяти» - важный частный случай этой проблемы.
- Решить ее можно, только изменив архитектуру вычислителя.

Высокопроизводительные вычислители нетрадиционной архитектуры: пространство решений.

	Монолитные	Гибридные
Программи-руемые	Шкафы Nvidia Tesla	МВС Экспресс (ИПМ РАН)
Реконфигу-рируемые	Продукция НИИ МВС (Таганрог)	Скиф-Аврора Cray XT5G,… (наш случай)

Гибридный суперкомпьютер: достоинства и недостатки.

- Основная работа по подготовке приложения к переносу – глубокая структурная перестройка алгоритма – выполняется в терминах традиционных фоннеймановских технологий.(**+**)
- Собственно переносу на нетрадиционный вычислитель подвергается **небольшая и простая** часть исходного текста – «вычислительные ядра».(**+**)
- НО!!! Очень высоки требования к коммуникационной системе (не зря первыми были Cray и SGI).(-)
- Поскольку сегодня эта проблема решаема, считаем, что плюсы перевешивают.
- Далее будем рассматривать технологии программирования вычислительных ядер для гибридных реконфигурируемых вычислительных систем общего назначения.

Важные исходные посылки.

- Современные САПР программируемой логики полностью изолируют схемотехнику (логику) от электроники (физики).
- Следовательно, базовой радиоэлектронной подготовки для создания схем в ПЛИС не требуется.
- Программируемая логика примерно в 20 раз медленнее жесткой.
- Следовательно, ускоряться менее, чем примерно в 100 раз в расчете на такт, не имеет смысла.

Так в чем же проблема?

- Видим 2 очевидных источника технологии разработки:

1). Привычные языки фоннеймановского типа (возможно, с какими-то расширениями),

2). Технологии, применяемые профессиональными схемотехниками (процессор одной задачи – просто частный случай схемы вообще).

- Почему-то все продолжают жаловаться на «очень высокую сложность и трудоемкость».

Источники ускорения расчета при схемной реализации. Непригодность фоннеймановских языков.

За счет чего достигается ускорение?

В общем – за счет переноса вспомогательной работы программы в структуру схемы («более глубокая компиляция»).

Конкретно

– за счет использования конвейерных арифметических устройств с большой задержкой, которые экономны в реализации, и, следовательно, их можно поставить много,

– за счет использования распределенной блочной памяти вместо единого запоминающего устройства, что позволяет многим конвейерам работать одновременно.

- Получается векторно – конвейерная синхронная машина.
- Вывод: картина мира для программиста радикально отличается от фоннеймановской. Нужны модели программирования «гораздо более другие», нежели в свое время потребовались для перехода от последовательных вычислений к параллельным. Иначе – просто замедлимся в 20 раз.

Как поступают профессионалы?

- Либо строят схему структурно, подобно тому, как раньше чертили на ватмане,
- Либо описывают ее на специальных языках программирования (VHDL и Verilog). Схемы получаются эффективные, НО:
- Языки эти безобразно спроектированы, пользоваться ими, только прочитав описание и не понимая, во что они транслируются, физически невозможно. Язык не формирует в голове программиста картину мира, а намеренно запутывает ее. Выживает тот, у кого эта картина сформирована до знакомства с языком, т. е. профессиональный схемотехник. Остальные – обречены.
- Нужен язык прикладного схемотехнического проектирования. Их сейчас многие активно делают. Что получается?

Что предлагается сегодня?

- Система Colamo решает другие задачи.
- Система Catapult C обещает невозможное (автоматическая трансляция обычного C++ прямо в схему).
- Системы Impulse C и Mitrion C вводят специфическую модель программирования и промежуточный уровень отображения ее в схему,
- Система Handel C предлагает модель программирования, прекрасно объяснимую в фоннеймановских терминах, очень эффективно отображаемую в схему, но на слишком бедное подмножество схемных конструкций (нет средств описать конвейер, что, как мы видели, принципиально важно).
- Налицо двойные потери: инструмент уже тяжел в освоении программистом, но еще недостаточно эффективен для оборудования.
- Результат (по экспресс – опросу общественного мнения): схема типичного простого модельного фрагмента, разработанная прикладным программистом при содействии профессионалов – схемотехников, работает медленнее качественной профессиональной реализации, выполненной силами схемотехников, примерно в 5 раз. Признать такой результат успехом трудно, есть еще над чем работать.

Что предлагаем мы?

- Модель программирования VHDL/Verilog надо «отжать», выразить в предельно простых и ортогональных терминах, а затем ускоренно пройти эмбриональный цикл фоннеймановских языков (ручное кодирование – ассемблер – BLISS/Астра – Фортан/C).
- Важное замечание: в приведенной эволюционной цепочке наибольший скачок в облегчении жизни программиста приходится на первый шаг: переход от ручного программирования к ассемблеру. А ведь модели программирования этот шаг не затронул, и **эффективности получающейся программы не снизил**.
- Следовательно, необходим «ассемблер» для модели VHDL/Verilog применительно к схемам вычислительного характера. Отсюда – название нашего языка – ***Автокод HDL*** («Автокодами 1 в 1» когда-то называли ассемблеры).

Упрощенная схемотехническая картина мира. Сравнение с фоннеймановской программой.

- Программа состоит из операторов. Она имеет пространственную структуру, но выполнение ее разворачивается **только во времени**. В каждый момент времени один оператор выполняется, остальные – пассивны.
- Схема состоит из соединенных между собой функциональных блоков. Выходы одних блоков соединяются со входами других, образуя пространственную структуру. В отличие от операторов программы, **все блоки всегда работают**, **вся** пространственная структура в **каждый** момент времени **активна**.
- Блоки имеют конечное время срабатывания. Чтобы это учесть, и не допустить распространения промежуточных, не установившихся состояний выходов на входы других блоков, некоторые блоки делают тактируемыми.
- Такие блоки воспринимают состояния своих входов только в моменты времени, задаваемые началом импульса тактового генератора. Тактовый генератор в схеме один, его выход разводится на специальные тактирующие входы всех таких блоков.
- Частота тактового генератора выбирается такой, чтобы вся логика, от которой непосредственно зависит любой вход любого тактируемого блока, успевала сработать за один такт. За этим, как и за отсутствием помех и наводок, следит САПР.

Упрощенная схемотехническая картина мира. Продолжение.

- Тактируемость придает работе схемы, наряду с пространственным, временнОе измерение, похожее на процесс выполнения программы. Можно говорить о блоках, «работавших» (изменивших значение выходов) и «не работавших» в данном такте, по аналогии с операторами программы, выполнявшимися или не выполнявшимися в данный момент времени.
- но с двумя отличиями:

- время дискретно,
- в каждый момент времени «работает» произвольное число блоков, а не один.
- Таким образом, отличие схемы от фоннеймановской программы сводится к следующему: ***программа выполняется в одном измерении – в непрерывном времени, ход которого задается ритмом срабатывания операторов, а схема выполняется в двух измерениях – в пространстве и в дискретном времени, ход которого задается тактовым генератором***.

Схемотехническая модель программирования.

- Не тактируемая логика естественно выражается в виде однократного присваивания (алгебраического равенства). Например: «**a = b + c**» означает: «пусть **a** будет всегда равно сумме **b** и **c**». Использование отдельно описанных (вложенных) блоков реализуется через вставку компонентов: «включить в схему блок типа **abc**, соединив его вход **d** с внешним сигналом **f**, а выход **g** – с внешним сигналом **x**». В программистских терминах это просто вызов макроса с ключевыми параметрами. Логически вся не тактируемая логика срабатывает мгновенно.
- Тактируемая логика естественно выражается в виде обобщенного фоннеймановского присваивания в дискретном времени, отложенного на такт. Например: «**a = b + c**» означает: «пусть на следующем такте **a** станет равно сумме значений **b** и **c,** взятых на текущем такте».
- Тактируемая и не тактируемая логика выписывается в разных разделах исходного текста.
- Текущей точкой выполнения схемы является группа тактируемых операторов присваивания, активных на данном такте. Все эти операторы выполняются одновременно и независимо.
- Каждый выполняемый оператор становится функциональным блоком схемы.

Схемотехническая модель программирования (продолжение).

- Условные конструкции присутствуют как в тактируемом, так и в не тактируемом случае, всегда выполняются мгновенно.
- Невозможность соединения выходов между собой естественно выражается в фундаментальном ***правиле единственности источника значения***.

Добавляем:

- раздел действий по начальному сбросу,
- раздел действий на каждом такте,
- **goto**, отложенный на такт,
- векторные конструкции и циклы периода компиляции,
- индексируемые массивы с регистрами доступа и однотактной задержкой чтения. Например:

  {
  myarray.addra = 1000
  myarray.dina = 15
  myarray.wea = 1
  }

  означает: «одновременно занести в регистр адреса массива **myarray** значение **1000**, в регистр входных данных – значение **15**, в регистр разрешения записи – единицу», то есть выполнить отложенное на такт присваивание «**myarray[1000] = 15**».

- Базовый язык готов. Он получился цикло-аккуратный и двумерный (пространство + время).

Замечания о реализации.

- Автокод транслируется в VHDL.
- Предусмотрен платформенно – независимый интерфейс схемы с управляющей программой.
- Базовый уровень языка подробно документирован, включая тривиальные примеры.
- Каждый пример включает в себя схему и запускающую ее программу, их взаимодействие подробно разбирается.

Схема копирования массива «d» в массив «c»:

```
program test                // Фиктивный заголовок
in 32 DI                    // внешнего интерфейса,
out 32 DO                   // чтобы транслятор
in 1 WE                     // не ругался
endprogram                  // конец заголовка
declare                     // Раздел объявления переменных
        ram 32 c(8, 320)    // int c[40][8]; // Массив восьмислойной памяти, все регистры доступа – вектора шириной 8.
        ram 32 d(8, 320)    // int d[40][8]; // Массив устроен так же.
        reg  9 i            // int i; // Переменная цикла – 9-битный регистр
enddeclare                  // конец объявления переменных
                            // Раздел комбинационной логики (однократных присваиваний) пуст
[                           // Раздел действий по начальному сбросу
         i = 0              // Счетчик цикла обнулить
         c.wea = 0          // Разрешение записи в «c» по порту «a» обнулить
         d.wea = 0          // То же для «d»
[]                          // конец раздела действий по начальному сбросу
                            // Раздел действий на каждом такте пуст
]                           // Раздел последовательных действий. Каждая пара фигурных скобок – один такт:
        {
         d.addra = 0        // Адрес доступа в «d» по порту «a» обнулить
        }
        {
         d.addra++          // Адрес доступа в «d» по порту «a» продвинуть (чтение происходит с задержкой на такт)
         c.addra = 0        // Адрес доступа в «c» по порту «a» обнулить
        }
loop1:                      // Цикл переписи восьмерок чисел
        {                   // Все тело цикла – один такт:
          d.addra++         // Продвинуть адрес для «d»
          if ( i < 40 )     // Проверить условие завершения
            next loop1      // Если не конец, на следующем такте перейти к началу
          endif
          c.addra++         // Продвинуть адрес для «c»
          c.dina = d.douta  // Присвоить очередную восьмерку
          c.wea = 1         // Запись в «c» разрешить
          i++               // Продвинуть счетчик цикла
        }                   // Конец тела цикла
        {
         c.wea = 0          // Копирование окончено, запись в «c» запретить
        }
```

То же на VHDL (1):

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library proc_common_v1_00_b;
use proc_common_v1_00_b.proc_common_pkg.all;
entity test is
 port (
DI: in std_logic_vector(31 downto 0);
DO: out std_logic_vector(31 downto 0);
WE: in std_logic_vector(0 downto 0);
Clk: in std_logic;
Reset: in std_logic
);
end test;
architecture IMP of test is
  component HI_RAMB
   generic (
         data_width: integer;
         address_width: integer;
         depth: integer
         );
   port (
         doa: out std_logic_vector(data_width-1 downto 0);
         dob: out std_logic_vector(data_width-1 downto 0);
         addra: in std_logic_vector(address_width-1 downto 0);
         addrb: in std_logic_vector(address_width-1 downto 0);
         clka: in std_ulogic;
         clkb: in std_ulogic;
         dia: in std_logic_vector(data_width-1 downto 0);
         dib: in std_logic_vector(data_width-1 downto 0);
         rsta: in std_ulogic;
         rstb: in std_ulogic;
         wea: in std_ulogic;
         web: in std_ulogic
         );
  end component;
         type l8w32_type is array (0 to 7) of std_logic_vector(31 downto 0);
         type l8w1_type is array (0 to 7) of std_logic_vector(0 downto 0);
```

То же на VHDL (2):

```
signal c_addra: l8w32_type;
        signal c_addrb: l8w32_type;
        signal c_dina: l8w32_type;
        signal c_dinb: l8w32_type;
        signal c_douta: l8w32_type;
        signal c_doutb: l8w32_type;
        signal c_wea: l8w1_type;
        signal c_web: l8w1_type;
        signal d_addra: l8w32_type;
        signal d_addrb: l8w32_type;
        signal d_dina: l8w32_type;
        signal d_dinb: l8w32_type;
        signal d_douta: l8w32_type;
        signal d_doutb: l8w32_type;
        signal d_wea: l8w1_type;
        signal d_web: l8w1_type;
        signal i: std_logic_vector( 8 downto 0 );
begin
c_gen: for c_i in 0 to 7 generate
begin
        c_inst: HI_RAMB
        generic map (
                        address_width => 32,
                        data_width => 32,
                        depth => 40
        )
        port map (
                        doa => c_douta(c_i),
                        dob => c_doutb(c_i),
                        dia => c_dina(c_i),
                        dib => c_dinb(c_i),
                        addra => c_addra(c_i),
                        addrb => c_addrb(c_i),
                        wea => c_wea(c_i)(0),
                        web => c_web(c_i)(0),
                        clka => Clk,
                        clkb => Clk,
                        rsta => Reset,
                        rstb => Reset
        );
end generate c_gen;
```

То же на VHDL (3):

```
d_gen: for d_i in 0 to 7 generate
begin
        d_inst: HI_RAMB
        generic map (
                        address_width => 32,
                        data_width => 32,
                        depth => 40
        )
        port map (
                        doa => d_douta(d_i),
                        dob => d_doutb(d_i),
                        dia => d_dina(d_i),
                        dib => d_dinb(d_i),
                        addra => d_addra(d_i),
                        addrb => d_addrb(d_i),
                        wea => d_wea(d_i)(0),
                        web => d_web(d_i)(0),
                        clka => Clk,
                        clkb => Clk,
                        rsta => Reset,
                        rstb => Reset
        );
end generate d_gen;
STATE_PROC: process ( Clk ) is
  variable stage: integer := 0;
begin
 if ( Clk'event and Clk = '1' ) then
  if ( Reset = '1' ) then
i <= "000000000";
for i14 in 0 to 7 loop
c_wea(i14) <= "0";
end loop;
for i15 in 0 to 7 loop
d_wea(i15) <= "0";
end loop;
else
case stage is
when 0 =>
  stage := stage + 1;
```

То же на VHDL (4):

```
for i19 in 0 to 7 loop
d_addra(i19) <= "00000000000000000000000000000000";
end loop;
when 1 =>
  stage := stage + 1;
for i22 in 0 to 7 loop
d_addra(i22) <= d_addra(i22) + "00000000000000000000000000000001";
end loop;
for i23 in 0 to 7 loop
c_addra(i23) <= "00000000000000000000000000000000";
end loop;
when 2 =>
  stage := stage + 1;
for i27 in 0 to 7 loop
d_addra(i27) <= d_addra(i27) + "00000000000000000000000000000001";
end loop;
if (i < "000101000") then
        stage := 2;
end if;
for i31 in 0 to 7 loop
c_addra(i31) <= c_addra(i31) + "00000000000000000000000000000001";
end loop;
for i32 in 0 to 7 loop
c_dina(i32) <= d_douta(i32);
end loop;
for i33 in 0 to 7 loop
c_wea(i33) <= "1";
end loop;
i <= i + "000000001";
when 3 =>
  stage := stage + 1;
for i37 in 0 to 7 loop
c_wea(i37) <= "0";
end loop;
when others => null;
    end case;
  end if;
end if;
end process STATE_PROC;
end IMP;
```

Расширения языка

- Добавлены фиксированные производные типы:

- стандартный заголовок (передача параметров между программой и схемой),
- конвейеризованные арифметико-логические выражения с плавающей точкой,
- задержка на N тактов,
- пятиточечный разностный шаблон.

- Работа с переменными расширенных типов записывается тем же способом, что и работа с переменными «ram»: синтаксически – через «поля структуры», по смыслу – через регистры доступа.
- Формат объявления этих переменных – особый.
- Работа с этими переменными транслируется в базовое подмножество отдельным проходом транслятора по тексту, то есть расширенные возможности сводятся к базовым и концептуально, и технически. Модели программирования расширения не затрагивают.
- Серьезная недоделка: средства расширения не встроены в язык. Пользователь не может писать свои расширения на базовом подмножестве. Это гораздо труднее сделать, чем на фоннеймановском языке, но вполне обозримо, и должно быть сделано.
- Использование расширений сжимает текст в несколько раз.

Что получается, а что нет.

- На базовом подмножестве получаются тривиальные примеры, вроде решения сеточного аналога задачи Дирихле методами Якоби и красно-черной релаксации. Из очевидных трудностей записи таких примеров появился первый слой расширений языка.
- На расширенном языке получилась задача Римана о распаде произвольного разрыва. Вычислительное ядро совпадает с выделенным при параллельной реализации на МВС Экспресс. Полезное быстродействие ядра на 16-миллионном кристалле – около 20 млрд. флопс с двойной точностью.
- Эффективность использования арифметических устройств определяется временем разгона и торможения конвейеров (около 250 тактов). Пропуски тактов в разогнанном конвейере отсутствуют, частота – 125 Мгц.
- Трудозатраты – около 2 человеко-недель. Объем исходных формул – 2 листа А4. Объем исходного текста схемы – 6КБ, объем сгенерированного кода на VHDL – 150КБ.
- Основные затраты времени – на отладку величины задержек, выравнивающих фазы потоков в конвейерах. Это – очевидное указание на то, каким будет следующий слой расширений языка.
- Время трассировки схемы – около 3 часов.
- Использовано около 70% арифметических устройств кристалла, около 30% универсальной логики.

Что дальше.

- Проблема языковой абстракции работы с вне-кристальной памятью большого объема.
- Проблема поиска общего языка (в прямом и переносном смысле) с ускорителями на базе видеоплат.
- В рамках поиска ее решения – попытка реализации конфигурируемой крупными блоками системы на базе синтезированных VLIW-процессоров, сравнение с уже реализованным подходом.

Спасибо за внимание.

Вопросы?