САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра системного программирования

Тодорук Евгений Анатольевич

Разработка и оптимизация времени исполнения алгоритмов обработки данных для многоядерного потокового процессора на ПЛИС

Бакалаврская работа

Допущена к защите.
Зав. кафедрой:
д. ф.-м. н., профессор А.Н. Терехов

Научный руководитель:
ст. преп. Б.Н. Кривошеин

Рецензент:
ст. преп., к. ф.-м. н., доцент Н.Ф. Фоминых

Санкт-Петербург
2013

SAINT-PETERSBURG STATE UNIVERSITY
Mathematics & Mechanics Faculty

Department of Software Engineering

Eugene Todoruk

Development and performance optimisation of data processing algorithms for multicore stream processor on FPGA

Bachelor's Thesis

Admitted for defence.
Head of the chair:
professor A.N. Terekhov

Scientific supervisor:
senior teacher B.N. Krivoshein

Reviewer:
senior teacher, Dr. N.F. Fominykh

Saint-Petersburg
2013

Оглавление

Введение

Развитие современных научных и производственных технологий приводит к стремительному росту объёмов информации, которую необходимо оперативно обрабатывать для получения результатов с минимальными временными задержками. К числу таких задач можно отнести обработку изображений в режиме реального времени, обработку мультимедийных видео потоков высокой чёткости в телевидении и многие другие. Постоянным ростом потоков данных характеризуются такие области, как геофизика (например, обработка данных скважинных измерений или сейсмического зондирования) и биоинформатика (анализ геномных последовательностей). Использование больших параллельных вычислительных систем становится невозможным, когда речь идёт о персональных компьютерах и мобильных устройствах. Например, в скважинной геофизике важно провести обработку в сжатые сроки в полевых условиях, когда высокопроизводительные многопроцессорные системы не доступны. Существующие методы реализации алгоритмов обработки данных с использованием универсальных и сигнальных процессоров зачастую не способны обеспечить обработку со скоростью поступления данных. Для персональных компьютеров существуют графические процессоры (GPU), которые уже позволяют в некоторых случаях производить вычисления для нужд пользователей, но для использования в мобильных устройствах они не подходят.

Во многих алгоритмах обработки (например, алгоритмы полного перебора) используются массивные однотипные вычисления. Такие алгоритмы могут исполняться в нескольких независимых потоках, что

позволит уменьшить общее время обработки. Процесс распараллеливания на несколько вычислительных ядер принято называть построением широкого вычислительного конвейера, состоящего из нескольких однотипных ветвей исполнения. В качестве вычислительных систем для параллельных вычислений сегодня используются либо многоядерные и многопроцессорные персональные компьютеры, либо кластерные системы на основе универсальных процессоров.

Современные микросхемы программируемой логики FPGA (Field-Programmable Gate Array) обеспечивают параллельное исполнение до сотен тысяч одновременных потоков, при этом объём внутренней памяти достигает десятков Мбит. FPGA являются программно - конфигурируемыми вычислителями, то есть связи между вычислительными примитивами и внутренней памятью задаются программистом. Такая система делает возможным построение вычислительной архитектуры, максимально соответствующей реализуемым алгоритмам. При этом микросхемы FPGA можно неограниченное количество раз перепрограммировать, что позволяет использовать одно аппаратное устройство для решения различных задач.

Осенью 2011 года стартовал студенческий проект, задачей которого стояла разработка многоядерного потокового процессора (МПП) на FPGA, ориентированного на потоковую обработку данных для встроенных применений. Архитектура параллельного вычислителя создавалась на базе управляющего процессора и массива простых вычислительных ядер на базе FPGA. На данный момент МПП реализован и его подробное описание можно прочитать в выпускной квалификационной работе Д.В. Солдатова "Разработка программно-аппаратной архитек-

туры многоядерного потокового процессора на ПЛИС" [1].

Целью данной работы является оценка эффективности работы алгоритмов на МПП по сравнению с работой алгоритмов на процессоре общего назначения.

1. Постановка задачи

Для достижения цели дипломной работы были выделены следующие задачи:

1. Изучить архитектуру многоядерного потокового процессора (МПП);
2. Изучить алгоритмы, поддающиеся распараллеливанию на данной архитектуре МПП;
3. Реализовать и оптимизировать время исполнения алгоритмов для МПП;
4. Реализовать алгоритмы на Си для Intel Core i7;
5. Протестировать работу алгоритмов на потоках данных больших объёмов;
6. Проанализировать результаты, полученные на разных архитектурах, и построить сводную таблицу результатов;
7. Оценить эффективность работы алгоритмов на МПП по сравнению с процессором общего назначения.

2. Многоядерный потоковый процессор

2.1. Структура

Многоядерный потоковый процессор (МПП) представляет из себя динамически коммутируемую сеть ядер (Рис. 1). Ядра объединены в слои, а слои в свою очередь объединены в сеть. Поток данных проходит конвейерную обработку от первого слоя к последнему. Благодаря слоям коммутации между слоями ядер можно задавать связи между ядрами соседних слоёв. Такая структура похожа на матричную сеть процессоров (Рис. 2) [2], за исключением того, что в ней нет связей между ядрами в одном слое, но зато есть динамическая связь между слоями.

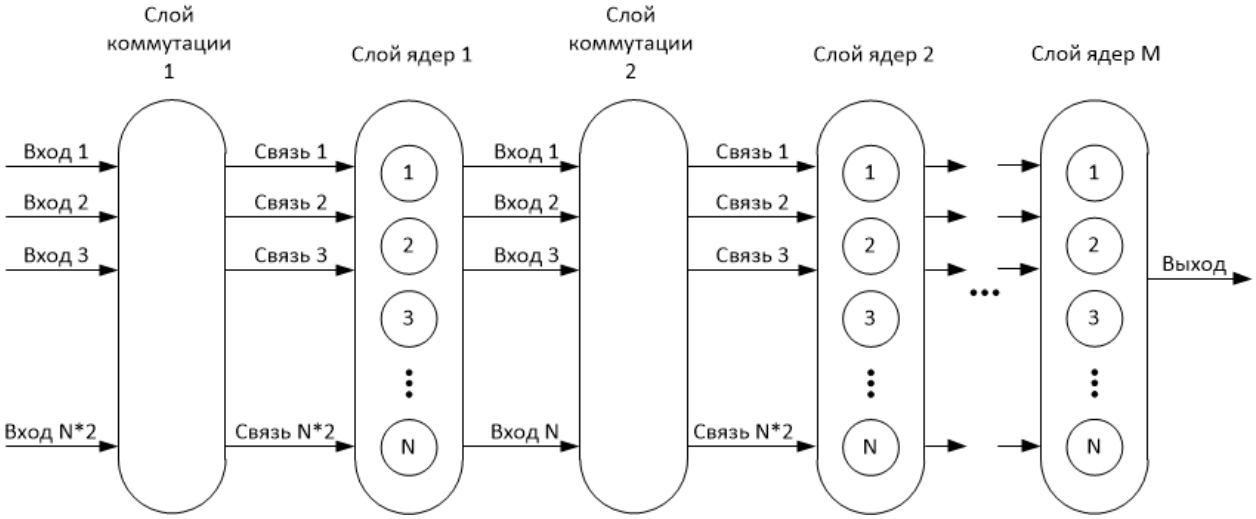

Рис. 1: Структура МПП

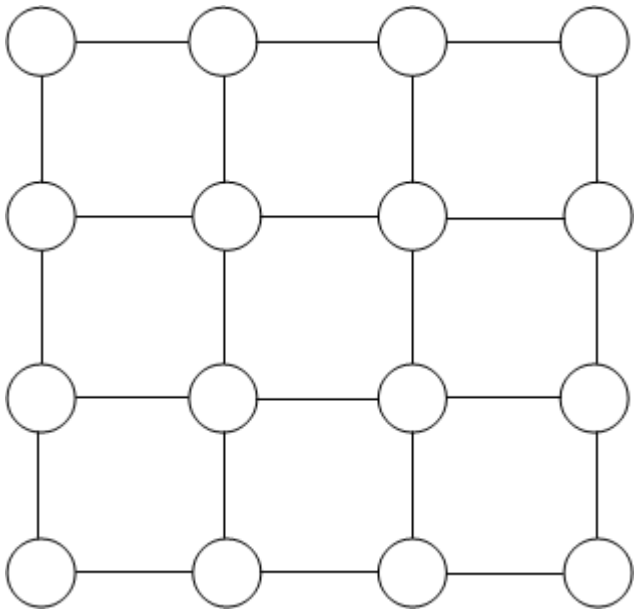

Рис. 2: Матричная сеть процессоров

2.2. Система команд МПП

Система команд МПП разделена на два типа:

1. команды ядра;

2. команды коммутатора.

Команды ядра

Ядро МПП - это модуль, предназначенный для управления вычислениями на ядре. Состоит из блоков ввода-вывода, памяти и цифрового сигнального процессора. Цифровой сигнальный процессор (англ. Digital signal processor, DSP) — специализированный микропроцессор, предназначенный для цифровой обработки сигналов в реальном масштабе времени. В FPGA Virtex-6 используется модуль DSP48E1 [3]., который позволяет выполнять операции вида $\pm(d \pm a)b - 1$, $c \pm (d \pm a)b$,

$\pm a : b \pm c - 1$ и т.п. за один такт. Микрокоманды для выполнения операций DSP получает из памяти, в которой хранится микропрограмма ядра. Структура микрокоманды отображена на Рис 3.

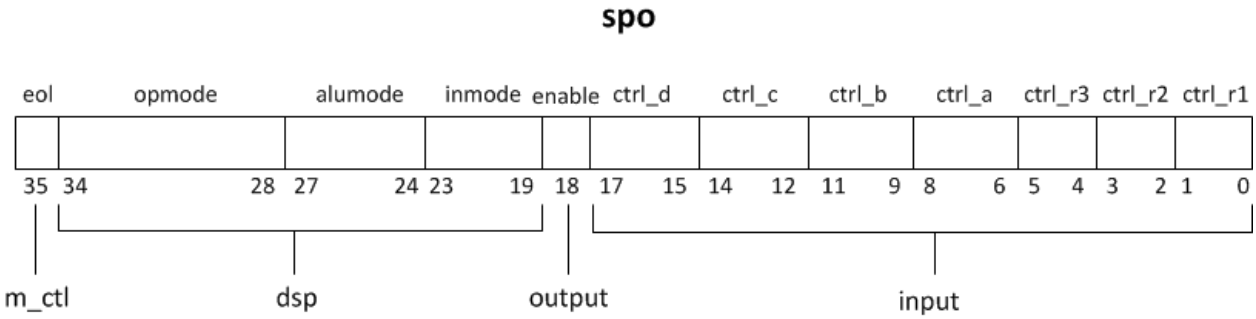

Рис. 3: Структура микрокоманды ядра

- `eol` — Флаг конца программы, 1 бит. Сигнал идёт на контроллер памяти, оповещая его, что программа выполнена;
- `opmode, alumode, inmode` — Параметры DSP, 16 бит. Этими командами задаются простые вычислительные операции;
- `enable` — Включение вывода, 1 бит. Этот сигнал подаётся, когда МПП готов выдавать результаты вычислений;
- `ctrl_d, ctrl_c, ctrl_b, ctrl_a, ctrl_r1, ctrl_r2, ctrl_r3` — Параметры входа, 18 бит. Используются для конфигурирования мультиплексоров.

Для программирования самого ядра были выделены следующие команды:

Код команды	Команда	Описание команды
000	BP	Пропускать входящие данные ”прозрачно”
001	WR1	Записать данные в регистр 1
010	WR2	Записать данные в регистр 2
011	WR3	Записать данные в регистр 3
100	WM	Записать программу в память
101	START	Начать выполнение программы из памяти
111	PAUSE	Приостановить выполнение программы

Данные и команды поступают в ядро независимо друг от друга, что позволяет программировать ядро ”на лету”.

Команды коммутатора

Как уже было сказано ранее, между слоями из ядер располагаются слои коммутации. Слой коммутации состоит из одного модуля коммутатора. У коммутатора имеется N входных портов для данных (по количеству ядер в сети) и 2N выходных портов данных, так как у каждого ядра два входных порта и один выходной. Для задания связей между ядрами используется команда коммутатора, состоящая из номера входного порта, номера ядра, которому предназначен вход, и номера входного порта ядра. Каждому входному порту соответствует хотя бы один выходной порт, а каждому выходному порту — не более, чем один выходной. На рис. 4 изображён пример связей коммутатора.

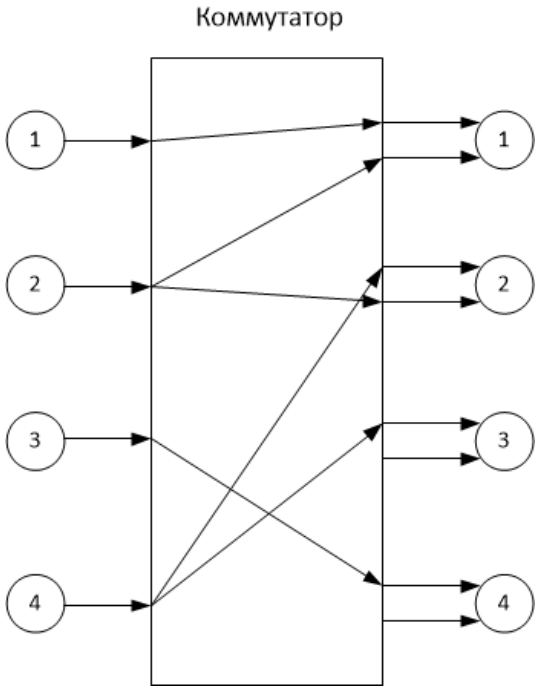

Рис. 4: Пример связей коммутатора

2.3. Программирование МПП

Обычно программирование МПП происходит в два этапа: программирование ядер и программирование связей между ядрами, после чего необходимо послать команду START для начала работы программы. Как уже было сказано в предыдущем пункте, данные и команды подаются независимо, и архитектура МПП позволяет его перепрограммировать в ходе выполнения программы.

Пакет МПП

Многоядерный потоковый процессор программируется посредством пакетов (Рис. 5).

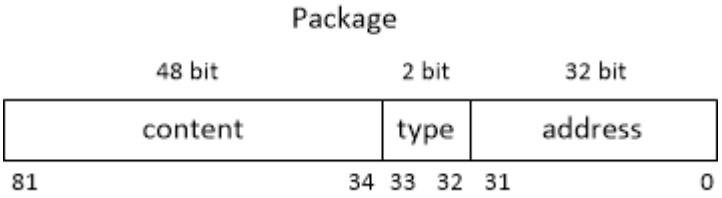

Рис. 5: Структура пакета МПП

- `content` — содержимое пакета, 48 бит. Может быть командой ядра или командой коммутатора в зависимости от типа пакета;
- `type` — тип пакета, 2 бита. 00 — команда ядра, 01 — команда коммутатора. Второй бит зарезервирован;
- `address` — адрес получателя, 32 бита. Адрес состоит из маски коммутаторов и маски ядер. По маскам определяются коммутаторы и ядра, которым был направлен пакет, благодаря чему мы можем одновременно программировать сразу несколько ядер.

Порты МПП

У МПП выведены наружу следующие порты:

- 2*N входных портов для данных (по два на каждое ядро в первом слое);
- 1 входной порт для программирования МПП;
- N выходных портов для результата вычислений.

3. Разработка программ на МПП

Программа на МПП — это последовательный набор пакетов и данных, где пакет содержит команды ядрам или коммутаторам, а данные — микропрограммы ядра. Так как работать с битовыми структурами крайне неудобно, были разработаны следующие две утилиты для упрощения программирования МПП.

Построитель микрокоманд ядра

Ранее уже была описана структура микрокоманды ядра (Рис. 3). Задача этой утилиты состоит в формировании микрокоманд. Она имеет консольный интерфейс, что позволяет без труда встроить её в в пользовательское приложение на языке более высокого уровня. Например, если мы хотим сложить данные, поступающих с обоих портов ядра и получить на выходе результат от сложения, то вызов утилиты будет выглядеть следующим образом:

Вызов утилиты из консоли с необходимыми параметрами:

```
$ ./spo_make -a i1 -d i2 -b r1 -inmode d+a -alumode z+x+y
-opmode m,m,0 -enable true -eol true
```

Сгенерированная микрокоманда на стандартном потоке вывода:

```
00000000000010000101000000100111100000111000000
```

Построитель пакетов МПП

Пакеты, как и микрокоманды ядра, имеют определённую битовую структуру. Эта утилита представляет из себя транслятор команд МПП. На вход транслятору подаётся текстовый файл с программой, и на выходе мы получаем битовую последовательность пакетов МПП и бито-

вый поток данных. Язык программы МПП состоит из одноимённых команд ядра и команды программирования коммутаторов SW. Следующий пример — это программа A+B на языке МПП:

```
# команда для коммутатора 1, связывает первый входной
# порт с первым входным портом ядра 1
SW 1 1 1 1
# команда для коммутатора 1, связывает второй входной
# порт со вторым входным портом ядра 1
SW 1 2 1 2
# команда для первого слоя и первого ядра. Записывает
# значение 1 в первый регистр
WR1 1 1 1
# команда для первого слоя и первого ядра. Записывает
# в память ядра микрокоманду для сложения значений на
# входных портах и вывода результата на выходной порт
WM 1 1 00000000000010000101000000100111100000111000000
```

4. Реализация алгоритмов

Архитектура МПП позволяет программировать сеть размера 16x18 (16 ядер в одном слое и 18 слоёв). Такие ограничения связаны с максимально допустимым количеством размещаемых модулей DSP и пропускной способностью основной шины на плате Virtex-6, под которую разрабатывалось устройство. Исходя из ограничений сети и задачи потоковой обработки данных для оценки эффективности МПП были выбраны следующие алгоритмы:

1. Фильтр Лапласа;
2. Свёртка;
3. Быстрое преобразование Фурье.

4.1. Фильтр Лапласа

Дискретный оператор Лапласа часто используется в обработке изображений, например в задаче выделения границ или в приложениях оценки движения. Дискретный лапласиан определяется как сумма вторых производных и вычисляется как сумма перепадов на соседях центрального пикселя. Для двухмерных сигналов дискретный лапласиан можно задать как свёртку со следующим ядром:

$$\mathrm{G} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Чтобы выполнить свёртку таким ядром нам потребуется всего 4 сложения и одно умножение. Для реализации на МПП достаточно трёх

слоёв ядер по три ядра в каждом слое. На рисунке 6 изображена структура сети МПП, по которой видно, что четыре ядра занимаются сложением, одно ядро умножением на константу, одно ядро пропускает данные ”прозрачно” и три ядра не задействованы.

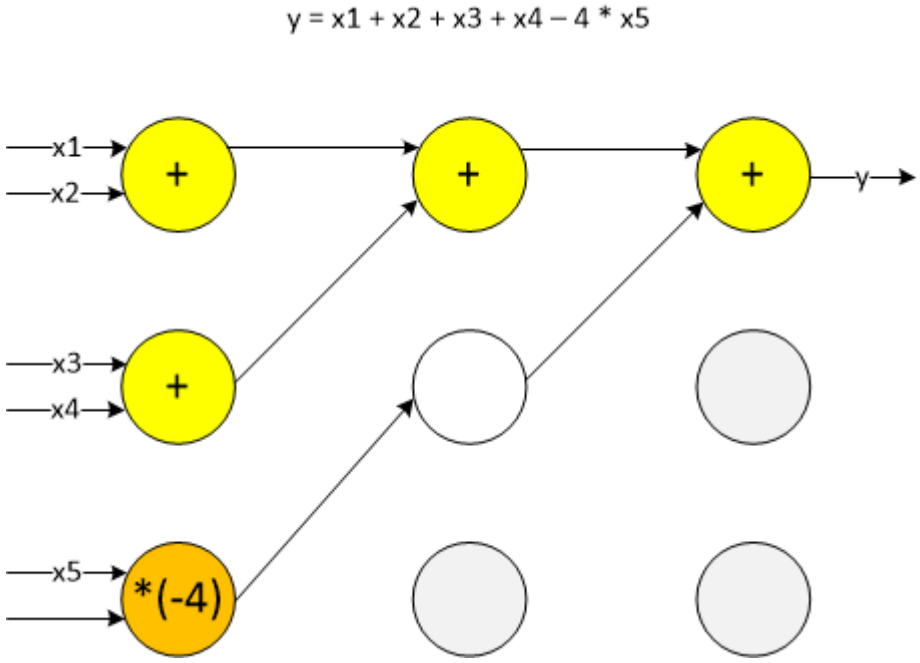

Рис. 6: Фильтр Лапласа на МПП

При такой структуре одна итерация алгоритма будет выполняться за один такт МПП. Для оптимизации времени исполнения алгоритма есть возможность увеличить количество итераций за такт МПП путём использования незадействованных ядер слоя. На данный момент задействованы 3 ряда ядер из 16 доступных и выполняется одна итерация алгоритма. Поэтому, задействовав 15 рядов ядер, мы обеспечим работу пяти итераций алгоритма за один такт МПП (Рис. 7). Такая оптимизация уменьшает время исполнения фильтрации Лапласа в 5 раз.

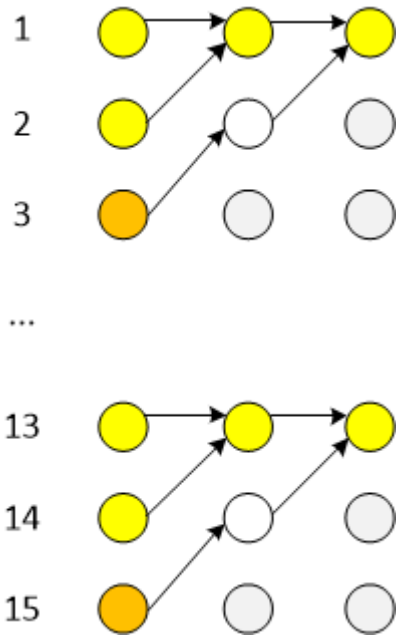

Рис. 7: Оптимизированный фильтр Лапласа на МПП

4.2. Свёртка

Свёртка — это основа большого количества трансформаций изображений. В каком-то виде мы уже применили свёртку в предыдущем алгоритме. Теперь рассмотрим более общий случай свёртки двухмерного сигнала. Так как сеть МПП ограничена, рассмотрим свёртку ядром фиксированного размера 10x10. Нам необходимо реализовать поэлементное умножение двух матриц 10x10 и сложение элементов результирующей матрицы:

$$\sum_{i=1}^{10}\sum_{j=1}^{10} x_{i,j} * a_{i,j}$$

где $a_{i,j}$ — элемент ядра свёртки. Для реализации нам потребуется сеть МПП 10x14 (Рис. 8): первые 10 слоёв МПП будут выполнять умножение на константу ядра свёртки и складывать с результатом, полученным на предыдущем слое, а последние 4 слоя — складывать оставшиеся резуль-

таты.

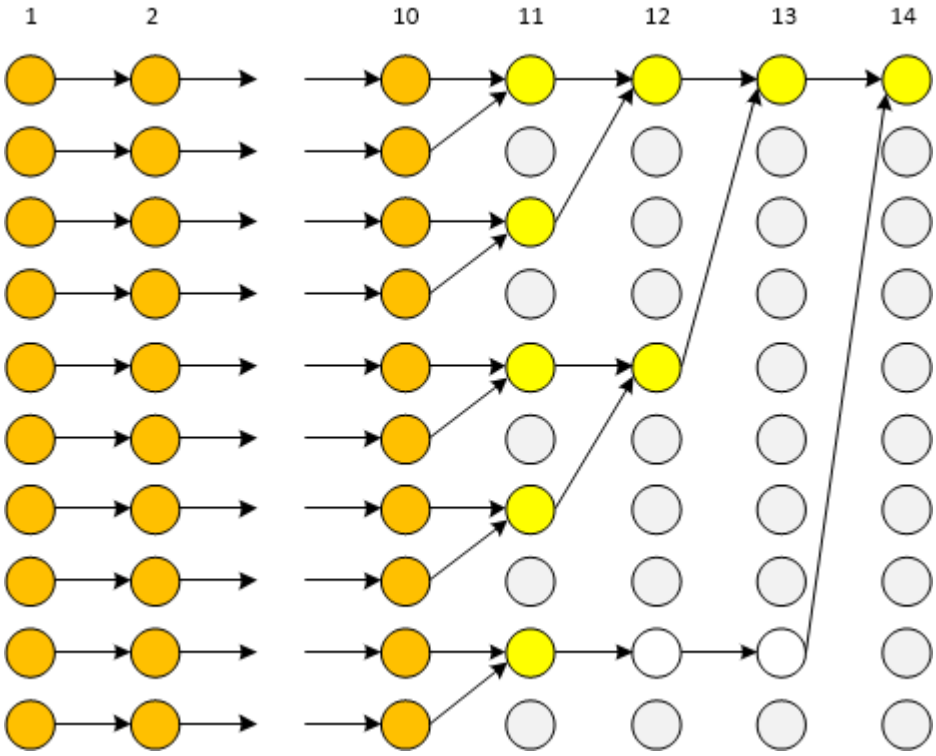

Рис. 8: Структура алгоритма свёртки на МПП

Примерный алгоритм работы каждого слоя сети МПП выглядит следующим образом:

Слой 1:

На первом такте умножить на константу и передать результат. На следующих 9 тактах передавать данные "прозрачно".

Слой 2:

На первом такте пропустить данные "прозрачно". На втором такте умножить на константу, сложить с предыдущим результатом и передать результат. На следующих 8 тактах передавать данные "прозрачно".

Слой 3:

На первых двух тактах пропустить данные "прозрачно". На третьем такте умножить на константу, сложить с предыдущим результатом и передать результат. На следующих 7 тактах передавать данные "прозрачно".

. . .

Слой 10:

На первых 9 тактах пропустить данные "прозрачно". На 10ом такте умножить на константу и передать результат.

Слой 11-14:

Сложить данные со входных портов и передать результат.

Таким образом свёртка ядром 10x10 будет занимать 10 тактов МПП. Чтобы уменьшить количество тактов на итерацию алгоритма, задействуем максимальное количество ядер в слое. Таким образом за один так мы будем получать 16 значений на вход вместо 10, и нам потребуется всего 7 слоёв для умножения на элемент ядра свёртки и 4 слоя для остаточных сложений. В итоге получим 7 тактов МПП на одну

итерацию алгоритма свёртки, что уменьшает время работы в 0,7 раз.

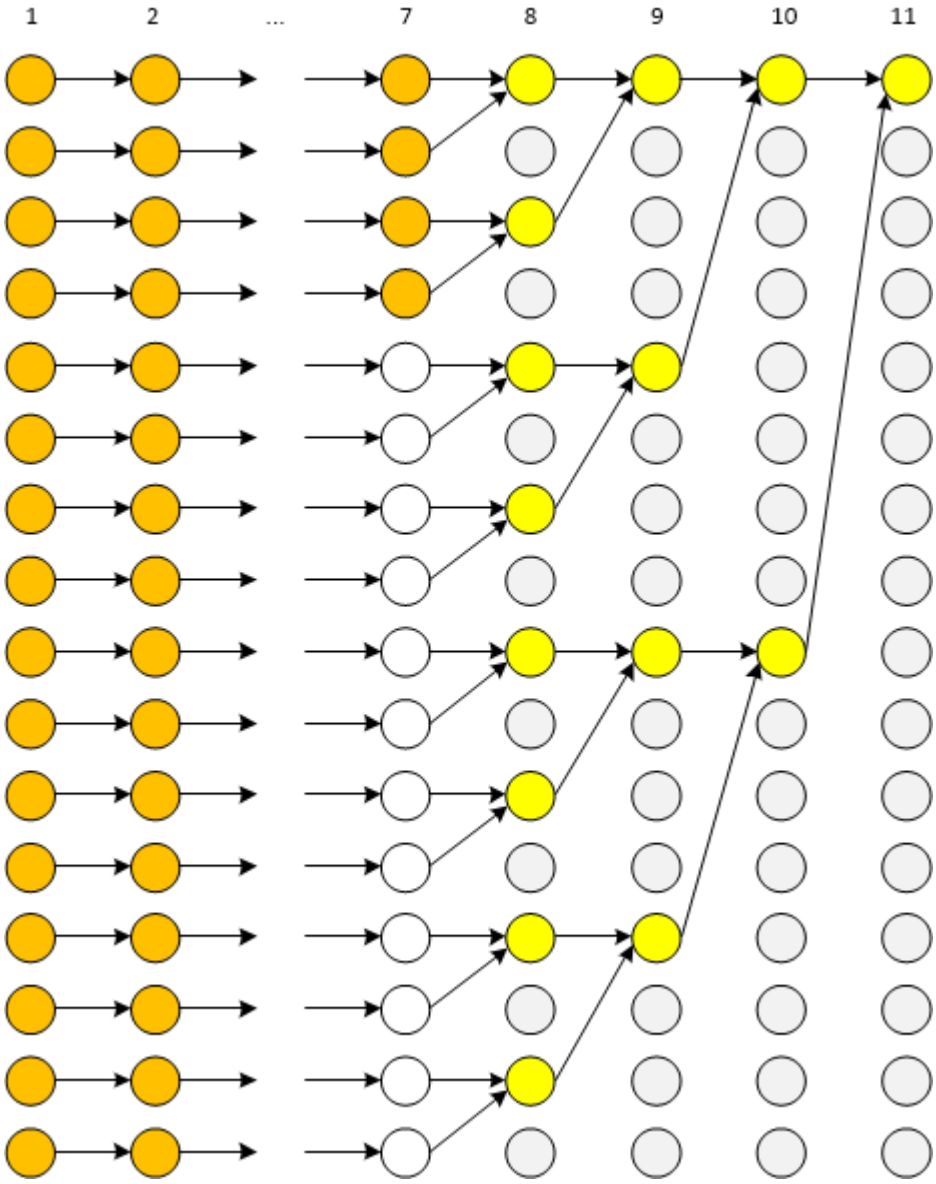

Рис. 9: Оптимизированный алгоритм свёртки на МПП

4.3. Быстрое преобразование Фурье

Быстрое преобразование Фурье (БПФ, FFT) — это алгоритм быстрого вычисления дискретного преобразования Фурье (ДПФ). Он широко применяется в алгоритмах цифровой обработки сигналов (его моди-

фикации применяются в сжатии звука в MP3, сжатии изображений в JPEG и др.), а также в других областях, связанных с анализом частот в дискретном сигнале. Вычисляется преобразование Фурье по следующим формулам:

Прямое преобразование:

$$X_k = \sum_{n=1}^{N-1} x_n e^{-\frac{2\pi i}{N} kn}, \quad k = 0, ..., N-1$$

Обратное преобразование:

$$x_n = \frac{1}{N} \sum_{k=1}^{N-1} X_K e^{\frac{2\pi i}{N} kn}, \quad n = 0, ..., N-1$$

Чаще всего используют приближённое дискретное преобразование Фурье, поэтому для реализации на МПП был взят один из самых популярных алгоритмов: БПФ по основанию 2 [4]. Общая структура одного шага БПФ даёт нам рисунок такого вида:

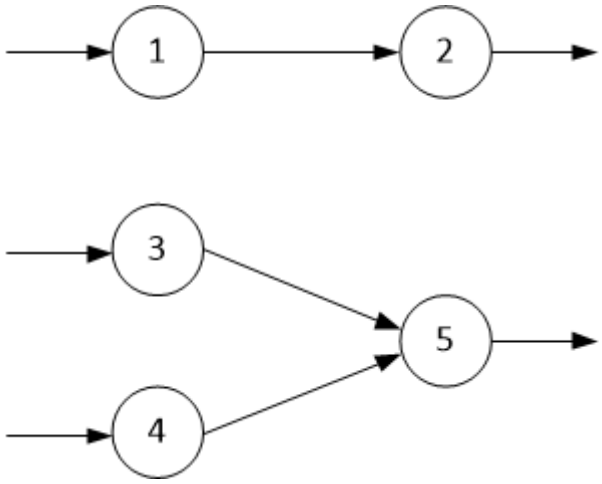

Рис. 10: Структура одного шага БПФ

На ядро с номером 1 подаётся первое комплексное число, на ядра 3 и 4 — второе, причём на первом такте подаётся вещественная часть

комплексного числа, а на втором — мнимая. Ядра 1 и 2 передают данные "прозрачно" с задержкой на один такт. Ядро 3 выполняет операцию $-RxRc+IxIc$, ядро 4 — $-RxIc-IxRc$, где Rx и Ix — это соответственно вещественная и мнимая часть входящего комплексного числа, а Rc и Ic — поворачивающий множитель преобразования Фурье. Задача ядра номер 5 состоит в том, чтобы последовательно выдать сначала результат, поступивший от ядра 3, а затем результат ядра 4. В итоге на выходе ядра 5 мы получим результат произведения входящего комплексного числа на поворачивающий множитель.

На рисунке 11 приведена общая структурная схема БПФ на МПП. На нём кругами обозначена макрооперация, описанная выше.

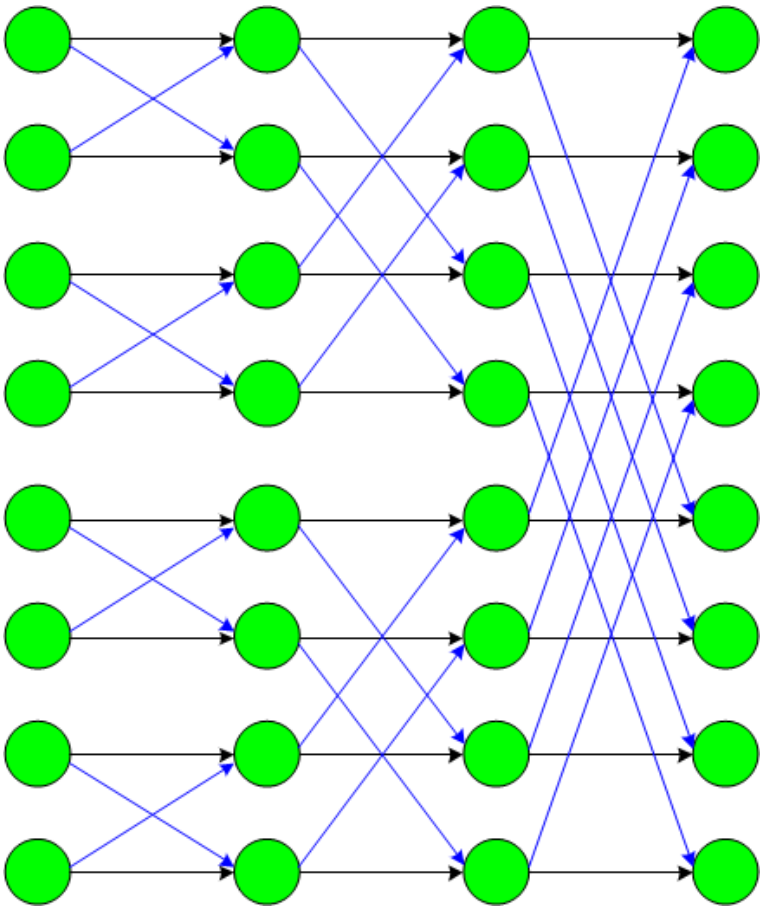

Рис. 11: Общая структурная схема БПФ 2^{16}

Таким образом одна итерация алгоритма занимает 4 такта МПП, причём оптимизировать время исполнения в данном случае невозможно, т.к. задействовано максимально допустимое количество ядер в одном слое.

5. Особенности реализации

Цифровой сигнальный процессор (DSP), который взят за основу вычислительного ядра МПП, предусматривает только целочисленные вычисления, а многие алгоритмы (например, БПФ) требуют работу с комплексными и вещественными числами. Для работы с такими числами были введены следующие способы представления чисел.

Вещественные числа

Так как арифметика чисел с плавающей запятой требует много тактов DSP, то было решено представлять вещественные числа в формате чисел с фиксированной запятой. На число в МПП приходится 30 бит, DSP позволяет делать 17 битный левый сдвиг, поэтому на целую часть числа с фиксированной запятой было выбрано 13 бит, а на дробную — 17 бит. Таким образом арифметика чисел в фиксированной запятой выполняется за два такта DSP.

Комплексные числа

Комплексные числа в МПП были представлены как пара вещественных чисел, где первое число — вещественная часть комплексного числа, а второе — мнимая часть комплексного числа.

6. Тестирование алгоритмов

Тестирование алгоритмов проходит по следующей схеме:

1. Подготовка тестового набора данных.

   На этом этапе мы подготавливаем данные, на которых будем тестировать алгоритм (например, множество изображений);

2. Формирование битового потока из тестового набора.

   Здесь мы из набора данных формируем структурированный битовый поток для тестируемого алгоритма;

3. Запуск алгоритма.

   Этот этап заключается в том, чтобы подать битовый поток данных на вход алгоритму и получить результат также в виде битового потока;

4. Формирование данных из битового потока.

   На последнем этапе мы из битового потока, полученного алгоритмом, восстанавливаем данные.

6.1. Тестирование на Intel Core i7

Для проведения вышеописанной схемы тестирования была разработана автоматическая система из bash скриптов. Формирование битовых потоков из данных и обратное формирование данных из потока осуществлено посредством MATLAB. Для получения результатов на Intel Core i7 алгоритмы были реализованы на Си с использованием библиотеки OpenMP [1] для параллельных вычислений.

[1] The OpenMP API specification for parallel programming. http://openmp.org/

Характеристики компьютера, на котором проводились испытания:
Процессор: Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz 8M Cache
Память: 8 ГБ
Операционная система: Ubuntu 12.04.

6.2. Тестирование на МПП

Тестирование алгоритмов на МПП проводилось в системе моделирования iSim [2], которая после синтеза логической структуры выполняет потактовое моделирование и даёт точное время работы алгоритмов. Для тестирования были написаны тестовые модули на языке VHDL, которые представляют из себя пару: генератор и анализатор. Генератор подаёт битовый поток данных на вход модулю МПП, а анализатор принимает выходной поток и проверяет его на корректность.

[2]ISE Simulator (ISim). http://www.xilinx.com/tools/isim.htm

7. Анализ результатов

Для первых двух тестируемых алгоритмов (фильтр Лапласа и алгоритм свёртки) на вход подавались потоки данных размера 98 и 137 Мбайт соответственно. Этот поток формировался из набора изображений. После работы алгоритмов мы получили набор обработанных изображений. При фильтрации Лапласа были получены изображения с выделенными границами, а при свёртке — размытые изображения (т.к. использовалось ядро Гаусса).

В случае алгоритма быстрого преобразования Фурье на вход подавался поток данных размера 65 Мбайт, состоящий из $256 * 2^{16}$ чисел типа float, сформированный случайным образом. Проверка результата работы алгоритма проходила в среде MATLAB посредством сравнения полученных данных с реальными результатами. Реальные результаты были посчитаны также в MATLAB при использования такого же массива данных при помощи встроенной функции `fft`.

Эффективность работы каждого алгоритма отображена в следующей сводной таблице. Последний столбец таблицы означает коэффициент эффективности.

Алгоритм	Поток (МБ)	Ширина потока (Б)	Время(с)		K
			Intel Core i7	МПП	
Фильтр Лапласа	98	25	4,7	0,009	522
Свёртка	137	100	1,715	0,0022	780
БПФ	65	64	1,526	0,0092	167

Стоит отметить, что результаты, полученные на Intel Core i7, за-

висят не только от частоты процессора и размера входного потока, а ещё и от таких факторов, как скорость работы с жёстким диском, тип операционной системы, используемый компилятор Си и так далее. Поэтому эти результаты надо считать приближёнными, так как они могут отклоняться при использовании разных конфигураций компьютера с одинаковым процессором, но ясно, что таких результатов, как на МПП, достичь не получится. Также можно заметить, что быстрое преобразование Фурье в случае Intel Core i7 работает быстрее алгоритма свёртки, а в случае МПП медленнее. Это связано с тем, что в МПП только целочисленная арифметика, и реализация арифметики вещественных и комплексных чисел выполнена программно, что существенно замедляет работу алгоритма.

Полученные результаты показали, что МПП хорошо справляется с потоковой обработкой данных, его архитектура больше всего подходит для реализации алгоритмов цифровой обработки сигналов и на поставленных задачах он работает в сотни раз быстрее, чем процессор общего назначения. Из возможных областей применения МПП можно перечислить следующие: кодирование аудио и видео потоков, распознавание речи и изображений, речевые и музыкальные синтезаторы, анализаторы спектра и многие другие области, где необходима быстродействующая обработка сигналов, в том числе в реальном времени.

Заключение

В ходе работы были выполнены основные цели:

1. Реализованы 3 алгоритма на МПП и на процессоре общего назначения Intel Core i7;
2. Произведена оценка эффективности работы алгоритмов на МПП по сравнению с работой алгоритмов на процессоре общего назначения Intel Core i7.

А также были решены дополнительные задачи:

1. Разработаны утилиты для упрощения программирования на МПП;
2. Разработаны утилиты для тестирования алгоритмов.

Список литературы

[1] Д.В. Солдатов. Разработка программно-аппаратной архитектуры многоядерного потокового процессора на ПЛИС. Бакалаврская работа, СПбГУ Математико-Механический факультет, 2013

[2] Р. Миллер, Л. Боксер. Последовательные и параллельные алгоритмы. Общий подход. БИНОМ, 2006

[3] Xilinx. Virtex-6 FPGA DSP48E1 Slice User Guide. UG369 edition, 2011.

[4] Лаборатория Параллельных информационных технологий НИВЦ МГУ. Основная схема быстрого преобразования Фурье.
URL: `http://fpga.parallel.ru/fft/`