

НЕКОТОРЫЕ АСПЕКТЫ ПРОГРАММИРОВАНИЯ ПРИКЛАДНЫХ АЛГОРИТМОВ ДЛЯ ПЛИС НА ЯЗЫКЕ COLAMO

А.В. Фролов

Использование ПЛИСов в настоящее время перестаёт быть прерогативой специалистов-схемотехников. Кроме языков низкого уровня, требующих составления детального расписания работы модулей, пользователи могут воспользоваться и языками высокого уровня. Таким языком высокого уровня, в частности, является Colamo, разработанный специалистами НИИ МВС ЮФУ. Его конструкции не требуют полного программирования всех микросхем ПЛИС, эту задачу выполняют транслятор и его среда. Поэтому программировать на этом языке могут и специалисты по прикладным алгоритмам, а не только схемотехники. Однако в реальности с достижением эффективности при программировании на Colamo могут быть сложности. О них, а также о некоторых других интересных аспектах языка Colamo, и идёт речь в докладе.

При трансляции написанной на Colamo программы пользователь получает довольно много информации из запускаемой в среде программы FireConstructor. В частности, он может реально проверить, какие именно ресурсы модулей ПЛИС будут использованы при работе, и как эти ресурсы соединяются друг с другом каналами передачи информации. Пользователю в принципе доступны и другие характеристики реально генерируемого кода, он может прочитать их в текстах разных частей программы на ассемблере Argus. Однако для “прикладника”, не являющегося специалистом-схемотехником, эти тексты будут “китайской грамотой”, а вот оценить, насколько хорошо использованы выделенные ресурсы, нужно.

В этой части нельзя считать, что, например, все задействованные устройства будут постоянно работать и что, таким образом, загрузка ПЛИСов можно просто посчитать, поделив количество этих ресурсов на полные ресурсы модулей. Например, в [1], где разобраны материалы семинаров НИИ МВС по языку, можно видеть на примерах рекурсивных алгоритмов, что для полной загрузки того же элементарного “сумматора” из библиотеки ядра, нужно, чтобы он обрабатывал конвейерный поток, в котором на разных стадиях суммирования находилось бы одновременно 18 пар чисел.

Для других элементарных операций количество ступеней другое. Здесь мы приведём краткую таблицу характеристик из библиотеки ядер для схем Virtex 5 и Spartan 3.

Операция	+/-	*	/	cos	sin	sqrt
FlipFlops	755	310	3449	2546	2573	341
LUTs	903	135	1528	2722	2722	355
RAMs	0	0	0	0	0	0
DSPs	0	4	0	0	0	0
Глубина конвейера	18	12	57	37	36	17

Как видно, разные операции захватывают разное количество ресурсов, и исполняются за разное количество тактов. Поэтому при разработке программы массовой обработки массивов какой-либо элементарной или составной операцией мы не можем писать такую программу «абстрактно», отвлекаясь от того, какая же это именно операция. Например, распространённая в прикладных вычислениях операция $a*b+c$ займёт 30 тактов, что делает невозможным использование для неё программы массового сложения с его 18 тактами на конвейер.

Однако в примерах из [1] даже не это самое проблемное. Оказывается, можно записать «хороший составной конвейер» из 4, 8, 16 сумматоров, но их работа проиграет суммированию на одном-единственном, если тот будет загружен максимально, а в «составном конвейере» всё будет заторможено обратной связью, которая даст использовать на каждом из отдельных сумматоров лишь одну из 18 его ступеней. Это хорошо видно при рассмотрении примеров с рекурсивными алгоритмами.

Все эти ситуации не видны из графической среды FireConstructor. Вопрос: что же делать? Так ли всё безнадежно? Нет. Если прикладной программист не будет полагаться при достижении эффективности на транслятор, а всё будет делать самостоятельно – эффективное программирование на Colamo вполне возможно. Для получения хороших эффективности, загрузки и ускорений на ПЛИСах с помощью Colamo просто надо проводить анализ информационного графа исполнения алгоритма непосредственно при программировании, а не после него. Учёт характеристик элементарных операций из библиотек при этом становится необходимым.

Теперь рассмотрим другой аспект погони за эффективностью — полноту загрузки имеющегося оборудования. В [2] мы видим, что в том же семействе микросхем Virtex 5 у нас очень широкий спектр возможностей. Рассмотрим самые мощные из микросхем этого типа – XC5VLX330T и XC5VSX240T. В первой из них заложено по 207360 устройств типа LUTs и FlipFlop, а также 192 устройства типа DSP48E, во второй — по 149760 LUTs и FlipFlop и 1056 DSP48E. Рассмотрим алгоритм, основной производительной частью которого

является операция $a*b+c$. Тогда каждая такая операция займёт 1065 FlipFlop, 1038 LUTs и 4 DSP48E (вопрос об их полноте загрузки пока оставим в стороне), выполняясь в конвейерном режиме за 30 тактов. Тогда схема XC5VLX330T сможет дать нам для выполнения только 68 таких фрагментов (больше - не хватает DSP), при этом останутся неиспользованными почти 2/3 устройств типа LUTs и FlipFlop, и вряд ли нам удастся «пристроить» их за счёт логических операций. В случае же схемы XC5VSX240T мы можем заполнить её 140 такими фрагментами, но при этом используем только 560 DSP, и почти половина из имеющихся на схеме устройств этого типа будет простаивать. В случае другого алгоритма такой дисбаланс и, следовательно, неполная загрузка оборудования, может быть больше или меньше, в зависимости от конкретных микросхем и от того, какую часть алгоритма мы доверяем исполнению ПЛИС. Отметим, что все эти рассуждения и вычисления — ещё при том предположении, что нам все эти операции удастся загрузить данными. Между тем, в разборе в [1] реализации на ПЛИС того же метода Гаусса отмечается, что параллельные возможности микросхем ограничены пропускной способностью их каналов ввода/вывода, и загрузить даже предоставленное транслятором с Colamo скорее можно только в конвейерном режиме. Тогда, если мы это попытаемся сделать для схемы XC5VLX330T, то выстраивание в конвейер всех 68 устройств типа $a*b+c$ даст нам конвейер более 2000 ступеней, что заставляет нас дать для его загрузки поток данных не менее чем в 2000 групп чисел, если нам нужна хотя бы половинная эффективность. Вычисление того, сколько чисел в каждой из таких групп, тоже поставит перед нами вопросы о каналах их подачи, и т.п.

Итак, мы видим, что, если у нас есть в вычислительной системе конкретные микросхемы, то вряд ли нам удастся выжать пиковую производительность или даже хотя бы приблизиться к ней, реализуя на ПЛИСе различные фрагменты. Ситуация может оказаться существенно лучше, если мы для конкретного алгоритма выбираем ту микросхему, которую потом вставим в реальное устройство. Именно этим и объясняется широкий спектр параметров в [2] и аналогичных описаниях семейств микросхем. Однако в случае, когда ПЛИСы используются как часть другой системы универсального назначения (типа кластера и т.п.) в качестве ускорителя и у прикладного пользователя нет возможности менять схемы, всё что он может — это варьировать ту часть алгоритма, которую он будет реализовывать на таких ПЛИС-ускорителях узлов кластера. В этой связи возникает математическая задача разложения части алгоритма по базису из тех операций, которые возможно «втиснуть» в реальные микросхемы. В ряде случаев прикладник может выбрать разные реализации одного и того же метода. Например, при реализации БПФ то же комплексное умножение возможно реализовать из разного количества действительных сложений и умножений. Выбор реализации должен зависеть от того, какую конкретную микросхему будет использовать программа.

Как видно из доклада, инструментов, имеющихся у прикладного математика, пишущего программы на языке Colamo, вполне достаточно для оценки того, какие ресурсы ПЛИС будут заняты задачей, но для оценки реально получаемых величины ускорения и эффективности распараллеливания наглядных средств может оказаться мало. Как и в случае с ЯПВУ традиционной конструкции, для получения нормальных ускорений прикладной программист должен хоть и не изучать все получаемые транслятором тексты на ассемблере с расписанными временными задержками, но, как минимум, исследовать получаемый при программировании граф исполнения алгоритма, включая параметры вершин этого графа. В этой связи следует добрым словом отметить то, что, несмотря на отчасти паскалеподобный стиль программ на Colamo, возможности, которые язык предоставляет программисту, напоминают скорее не заслоняющий «железо» от программиста Паскаль, а старый добрый Фортран. В особенности отметим то обстоятельство, что тот же принцип однократного присваивания, который в других языках скорее используется для повышения уровня абстрагирования от конкретных «железок», в Colamo как раз обусловлен требованиями, исходящими от оборудования, а именно ограничениями контроллера распределённой памяти. Поэтому и его выполнение просто необходимо для достижения высокой эффективности работы ПЛИСов.

ЛИТЕРАТУРА:

1. <http://colamo.parallel.ru>
2. http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf