УДК 004.272.44

ПРИМЕНЕНИЕ ПОТОКОВЫХ ВЫЧИСЛИТЕЛЬНЫХ МОДЕЛЕЙ В ПРОЕКТИРОВАНИИ СПЕЦИАЛИЗИРОВАННЫХ ПРОЦЕССОРОВ

Р.И. Попов

Представлен подход к проектированию специализированных процессоров на основе потоковой модели алгоритма. Рассмотрены основные классы потоковых моделей. Предложена модель с распределенным управлением, эффективная и простая в аппаратной реализации. Рассмотрены основные факторы, ограничивающие производительность потоковых процессоров.

Ключевые слова: потоковые процессоры, синтез специализированных процессоров, потоковые вычислительные модели, булевский поток данных.

Введение

Продолжающееся долгие годы эволюционное развитие полупроводниковых технологий сопровождается постоянным увеличением числа транзисторов в микросхемах и снижением их цены. Эффективное использование этих новых ресурсов для достижения большей производительности и энергоэффективности конечных решений с каждым годом становится все более сложной задачей. Традиционные архитектурные решения не всегда хорошо масштабируются, и проектировщикам приходится искать новые

подходы к организации вычислительного процесса в цифровых микросхемах. Одним из таких подходов является синтез специализированных процессоров из потоковой модели алгоритма.

Современные процессорные ядра общего назначения, как правило, построенные на базе суперскалярной архитектуры, оказываются очень эффективными в задачах с большим количеством ветвлений (за счет использования блока предсказания переходов) и в тех случаях, когда удается добиться высокого уровня параллелизма на уровне машинных команд. Однако, когда приходится иметь дело с обработкой больших объемов данных в реальном времени, производительность традиционных архитектур оказывается недостаточной или достигается за счет радикального повышения тактовой частоты при использовании высокоскоростных техпроцессов с большими токами утечки [1]. К таким задачам относятся реализация коммуникационных протоколов, видео- и аудиообработка, компьютерная графика и множество других задач из разных прикладных областей. Характерным для таких задач является применение набора однородных вычислений к большому объему данных в памяти (к примеру, сжатие данных) или к непрерывно поступающему потоку данных (к примеру, обработка аудиопотока). Наиболее эффективно такие задачи решаются на специализированных процессорах, использующих пространственный параллелизм, заложенный в алгоритмах. Такие процессоры, часто называемые потоковыми процессорами, аппаратно реализуют одну из потоковых вычислительных моделей.

Алгоритм для потокового процессора можно представить в виде графа потока данных (dataflow graph, DFG). DFG описывает алгоритм в виде направленного графа, где вершины представляют вычисления (функции), а дуги описывают потоки данных (коммуникации между вершинами). DFG не содержит информации о том, в каком порядке должны запускаться вычисления в вершинах и каким способом передаются данные, эти правила (семантика) задаются потоковой вычислительной моделью. Первой такой моделью считается модель сети процессов, предложенная Каном в 1974 г. [2]. За прошедшие 40 лет было предложено множество потоковых моделей, нашедших применение в проектировании и программировании систем цифровой обработки данных. В работе рассмотрены те из них, которые пригодны для прямой аппаратной реализации.

Процесс проектирования специализированного потокового процессора (акселератора) можно разбить на следующие шаги:

1. разработка и верификация алгоритма на императивном языке высокого уровня, к примеру, на C++ или в среде Matlab;
2. построение и оптимизация модели алгоритма в виде DFG;
3. реализация модели в виде специализированного процессора.

В настоящее время шаги 2 и 3 могут быть частично или полностью автоматизированы. Над автоматизацией и оптимизацией этого процесса работают несколько исследовательских коллективов по всему миру. Цель этой работы – рассмотреть выразительные возможности различных потоковых вычислительных моделей и выделить наиболее пригодную для реализации в аппаратуре.

Потоковые модели вычислений

Потоковые модели можно условно разделить на 2 класса: статически диспетчеризуемые (statically schedulable), для которых можно заранее (до запуска модели) составить расписание запуска вычислений, и динамические, для которых запуск тех или иных вычислений зависит от потока данных, поступающего в процессе работы. Динамические модели плохо подходят для реализации в аппаратуре, так как требуют динамического выделения буферов для передаваемых в процессе работы вычислительной системы данных. Для статически диспетчеризуемых моделей можно определить размер буферов между вычислительными узлами заранее, в процессе проектирования вычислительной системы.

Классической статически диспетчеризуемой моделью является модель с синхронным потоком данных (synchronous data flow, SDF) [3]. В SDF вычислительный узел потребляет и производит заданный заранее объем данных при каждом запуске. Эта информация позволяет рассчитать расписание запуска вычислений и размеры FIFO-буферов между узлами. Другой важной особенностью модели SDF является возможность приведения любой мультичастотной системы к моночастотной, как показано на рис. 1.

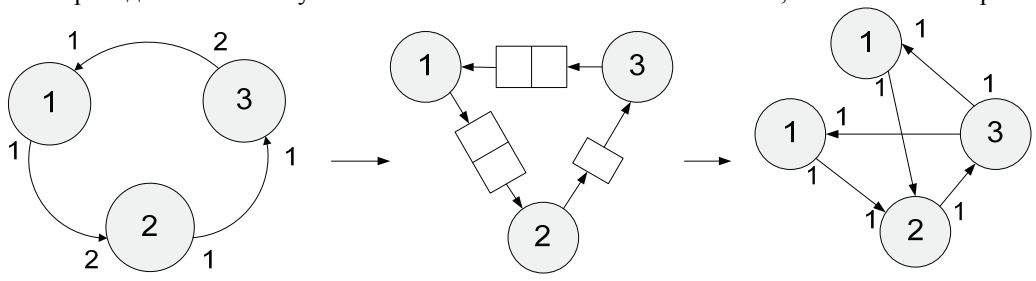
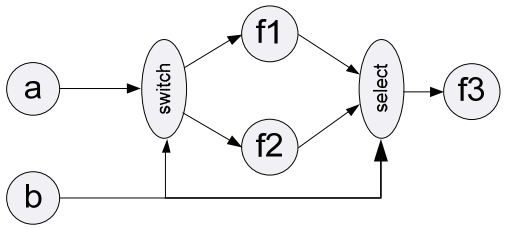


Рис. 1. Выделение FIFO-буферов в SDF модели и преобразование модели к моночастотной

Основным недостатком классической SDF модели является невозможность реализации ветвлений и циклов с переменным числом итераций. Узел, осуществляющий вычисления по условию, нарушает правила SDF, так как объем данных, производимых им, зависит от значения условия на входе.

Для реализации циклов и ветвлений для статически диспетчеризуемых моделей придумано несколько расширений, наиболее часто используемым из которых является булевский поток данных (Boolean Dataflow, BDF) [4]. Эта модель расширяет SDF двумя специальными узлами – SWITCH и SELECT, реализующими ветвления по булевому признаку. Пример реализации ветвления в BDF показан на рис. 2.




```
if (b)
     res =  f1(a);
else
     res = f2(a);
return f3(res);
```

Рис. 2. Реализация ветвлений в BDF

В классических трудах, посвященных потоковым вычислениям, в качестве их приложения рассматривается задача программирования многопроцессорных систем. Предполагается, что вычисления в каждом узле занимают значительное время. Для каждой модели составляется расписание запуска вычислений, максимально эффективно использующее те процессорные ресурсы, которые доступны разработчику.

В отличие от реализации на многопроцессорной системе, прямая аппаратная реализация позволяет полностью реализовать граф вычислений в виде вычислительного конвейера, тем самым отпадает необходимость создавать модуль, реализующий исполнение расписания вычислений. Это также упрощает процесс трассировки межсоединений в микросхеме, так как большинство связей становятся локальными. Однако в этом случае система будет производить неправильные данные до того момента, пока конвейер не будет полностью прогружен. Чтобы избежать этого, можно добавить признак достоверности ко всем данным, передающимся по системе. В таком случае вычислительный узел, получивший на входе данные без признака достоверности, на выходе также должен производить недостоверные данные, а узлы с побочными эффектами, такие как порты памяти, должны игнорировать такие данные.

Подобную модель можно считать расширением BDF. Такое расширение также упрощает реализацию ветвления, как показано на рис. 3. В этом случае узел SELECT передает на выход данные, пришедшие с признаком достоверности.

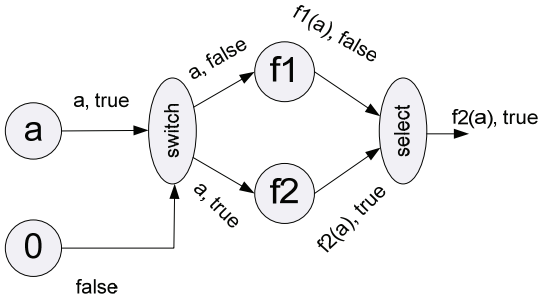


Рис. 3. Реализация ветвления в расширенной модели BDF

Еще одним полезным применением признака достоверности данных является его использование для отключения тактирования (clock gating) регистров в неактивных ветвях графа. Это позволяет значительно снизить динамическое энергопотребление в процессорах, реализующих алгоритмы с большим количеством ветвлений.

Далее рассмотрим на примере особенности реализации потоковой машины, реализующей данную вычислительную модель.

Аппаратная реализация потоковой машины

Вычислительную машину можно представить в виде тракта данных, состоящего из набора функциональных блоков различного назначения:

- вычислительных блоков, реализующих различные арифметические операции;
- портов ввода-вывода, используемых для получения данных из памяти (порты с адресом) и из различных каналов передачи данных (порты без адреса);
- конвейерных регистров.

Каждый функциональный блок может содержать от одного и более портов данных. Вычисления могут производиться с задержкой, укладывающейся в период тактового сигнала, или за несколько тактов. Пример функционального блока (двухстадийный умножитель) и его аппаратная реализация показаны на рис. 4.

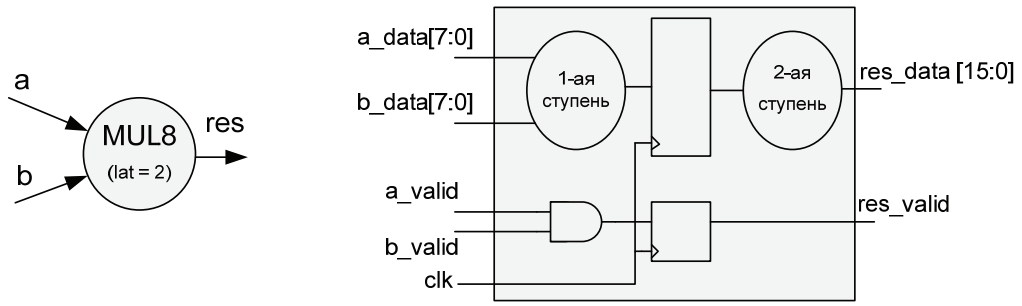


Рис. 4. Пример функционального блока – двухстадийный умножитель 8×8

В качестве примера акселератора рассмотрим модуль, вычисляющий следующую функцию:

```
void func (unsigned r, int *memory)
{
       for (i=0; i<r ; i++)
         if (i < r/2) memory[i] = memory[i] + i;
         else memory [i] = i;
}
```

Будем считать, что операция доступа к памяти занимает один такт, а все остальные операции выполняются в виде комбинационных схем и укладываются в период тактового сигнала. Модель одной из возможных реализаций этой функции показана на рис. 5.

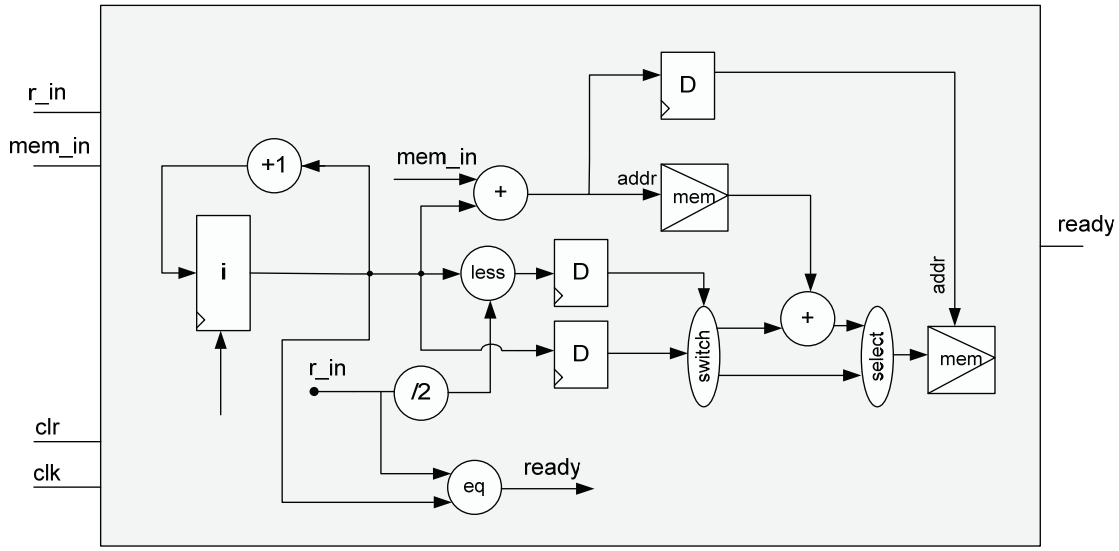


Рис. 5. Модель потокового акселератора

На схеме функциональные блоки комбинационного типа обозначены кругами, а функциональные блоки с внутренним состоянием и конвейерные регистры обозначены прямоугольниками.

Производительность потоковых акселераторов зависит от трех основных параметров.

1. Архитектура памяти. В случае, когда модель содержит несколько портов в общее адресное пространство, архитектура памяти (алгоритм работы кэша, количество банков и блоки предвыборки) должны быть выбраны таким образом, чтобы обеспечить бесконфликтное обращение по всем портам. В случае возникновения конфликта, исполнение модели останавливается до его разрешения.
2. Межитерационные зависимости по данным, ограничивающие возможности по распараллеливанию итераций циклов. Если *i*-я итерация цикла зависит от данных, получаемых на (*i*–1)-й итерации, это означает, что цикл не может быть полностью распараллелен.
3. Задержки на обратных связях. Предельно допустимая частота работы акселератора будет определяться максимальной задержкой на обратных связях. Во всех остальных случаях критические пути в схеме можно разбить конвейерными регистрами.

В показанном примере используются два порта в общее адресное пространство. Для обеспечения бесконфликтной работы памяти достаточно применить двухпортовую память или регистровый файл. В случае, когда доступна только однопортовая память, акселератор будет проводить половину времени работы в ожидании данных от памяти. В более сложных случаях для обеспечения высокой производительности алгоритм должен быть переработан для использования нескольких независимых банков памяти.

В показанном примере нет межитерационных зависимостей, поэтому он может быть легко распараллелен путем дублирования аппаратуры. Единственная обратная связь проходит через функциональный блок (+1) в счетчике итераций. Если задержка через этот функциональный блок составляет 5 нс, то акселератор сможет работать на частоте до 200 МГц при соответствующей конвейеризации остальной части тракта данных. Чтобы поднять частоту акселератора, потребуется разбить операцию на обратной связи на несколько стадий, однако это не приведет к увеличению производительности, так как счетчик итерации будет производить действительные данные только 1 раз за N тактов, где N – число конвейерных стадий на обратной связи.

Заключение

В работе представлены основные типы потоковых вычислительных моделей, пригодных для аппаратной реализации. Показан пример создания акселератора на основе расширенной BDF модели, являющейся одной из наиболее простых и эффективных при прямой реализации в аппаратуре. Выделены основные факторы, влияющие на производительность потоковых машин.

Потоковые модели обладают значительной выразительной мощью, что позволяет с их применением описывать широкий класс алгоритмов из различных предметных областей. При этом полученные модели пригодны как для прямой аппаратной реализации, так и для отображения на различные реконфигурируемые машины [5]. Наиболее перспективной областью применения потоковых моделей является их использование в виде промежуточного представления в системах высокоуровневого синтеза. С их помощью можно выразить все вычислительные и управляющие конструкции, используемые в современных языках высокого уровня, при этом они позволяют максимально использовать возможности распараллеливания вычислений в специализированной аппаратуре.

Литература

1. Budiu M., Artigas P.V., Goldstein S.C. Dataflow: A Complement to Superscalar // Proc. of the IEEE International Symposium on Performance Analysis of Systems and Software. – Washington, DC, USA, 2005. – P. 177–186.
2. Kahn G. The semantics of a simple language for parallel programming // Proc. of IFIP Congress. – Stockholm, Sweden, 1974. – P. 471–475.
3. Edward A. Lee, David G. Messerschmitt. Synchronous Data Flow // Proc. of the IEEE. – 1997. – V. 75. – № 9. – P. 1235–1245.
4. Buck J.T. A dynamic dataflow model suitable for efficient mixed hardware and software implementations of DSP applications // Proc. of the Third International Workshop on Hardware/Software Codesign. – France, 1994. – P. 165–172.
5. Swanson S., Michelson, K. WaveScalar // Proc. of the 36th annual IEEE/ACM International Symposium on Microarchitecture. – USA, 2003. – P. 291–302.

Попов Роман Игоревич – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, аспирант, ripopov@gmail.com