Методы адаптации параллельной потоковой вычислительной системы под задачи отдельных классов

А.Л. Стемпковский, А.В. Климов, Н.Н. Левченко, А.С. Окунев

Аннотация. В статье описывается архитектура параллельной потоковой вычислительной системы и основные ее параметры, изменяя которые, можно адаптировать ее под тот или иной специфический класс задач. Показывается, в частности, что выбор семейства аппаратных базисных функций распределения вычислений позволяет лучше приспосабливать систему к задачам с различными типами локализации вычислений. Приводится общая методология разработки специализированных параллельных вычислительных систем на основе потоковой модели вычислений с динамически формируемым контекстом. На примере конкретной задачи (БПФ) показаны результаты применения предложенных методов.

Ключевые слова: модель вычислений с управлением потоком данных, параллельные вычислительные процессы, адаптация вычислительной системы, функции распределения, локализация вычислений.

Введение

В Институте проблем проектирования в микроэлектронике Российской академии наук (ИППМ РАН) ведется разработка новых принципов организации вычислительного процесса и на их основе проектируется вычислительная система, использующая модель вычислений, управляемых потоком данных, - параллельная потоковая вычислительная система (ППВС). Как уже отмечалось в работе [1], в настоящее время требуется создание масштабируемых вычислительных систем с высокой реальной производительностью на задачах, имеющих практическое значение. Другими словами, актуально решение проблемы создания вычислительной системы, которая обеспечивала бы:

- снижение накладных расходов на распараллеливание вычислительных процессов;
- синхронизацию по данным в динамике вычислений, обеспечивающую высокую производительность при работе с разреженными и сложно организованными данными.

Такая вычислительная система с нетрадиционной архитектурой, с управлением потоком данных как раз и обладает свойствами, позволяющими решить эти проблемы. Работы по созданию систем, использующих принцип "потока данных" (dataflow), интенсивно проводились в Англии, США и Японии в 80-х - 90-х годах прошлого века. В настоящее время эти поисковые работы сосредоточены, в основном, в университетах Японии и США. Однако в этих разработках не был решен целый ряд принципиальных вопросов, сдерживающих развитие данного направления. Из современных проектов, в модели вычислений которых используется принцип потока данных, можно назвать «WaveScalar», «Merrimac», «TRIPS».

Архитектура параллельной потоковой вычислительной системы благодаря масштабируемости и аппаратной балансировке загрузки блоков системы способна решить проблему создания семейства процессоров на отечественной элементной базе для встроенных систем, настольных систем, рабочих станций, корпоративных серверов, систем реального времени и суперкомпьютеров.

Важно отметить, что модель вычислений с динамически формируемым контекстом позволяет реализовывать (адаптировать) архитектуру ППВС под выполнение конкретных классов задач (например, задач обработки изображений, численных методов и др.), а также создавать архитектуру в виде универсальной вычислительной системы.

Архитектура параллельной потоковой вычислительной системы

ППВС представляет собой многоядерную масштабируемую вычислительную систему (Рис. 1).

Между ядрами в системе передаются единицы информации, называемые токенами. Токен представляет собой структуру, содержащую данное (операнд), ключ, кратность, маску и набор служебных полей. Коммутация между ядрами осуществляется на основе значения номера ядра, вырабатываемого блоком хеширования на основе настраиваемой функции распределения.

Ядро вычислительной системы (Рис. 2) конструктивно состоит из:

- модуля ассоциативной памяти (МАП), в котором происходит сопоставление токенов по определенным правилам и формирование пакетов (структуры данных, которая содержит операнды, необходимые для активации программы узла);
- исполнительного устройства (ИУ), в котором в соответствии с программой узла происходит обработка пакета и генерация новых токенов;
- блока хеш-функции, в котором осуществляется определение номера ядра, в которое передается токен из исполнительного устройства;
- внутреннего коммутатора (ВКМ), который передает токен либо во внешнюю сеть, либо в свой МАП.

Архитектура ППВС масштабируема и при увеличении числа ядер в системе падение реальной производительности на задачах со сложно организованными данными происходит существенно медленнее, чем при решении подобных задач на вычислительных системах с классической архитектурой. Это достигается, во-первых, за счет использования принципа потока данных, когда готовые к выполнению данные активируют выполнение программы узла; во-вторых, активируемому узлу для пол-

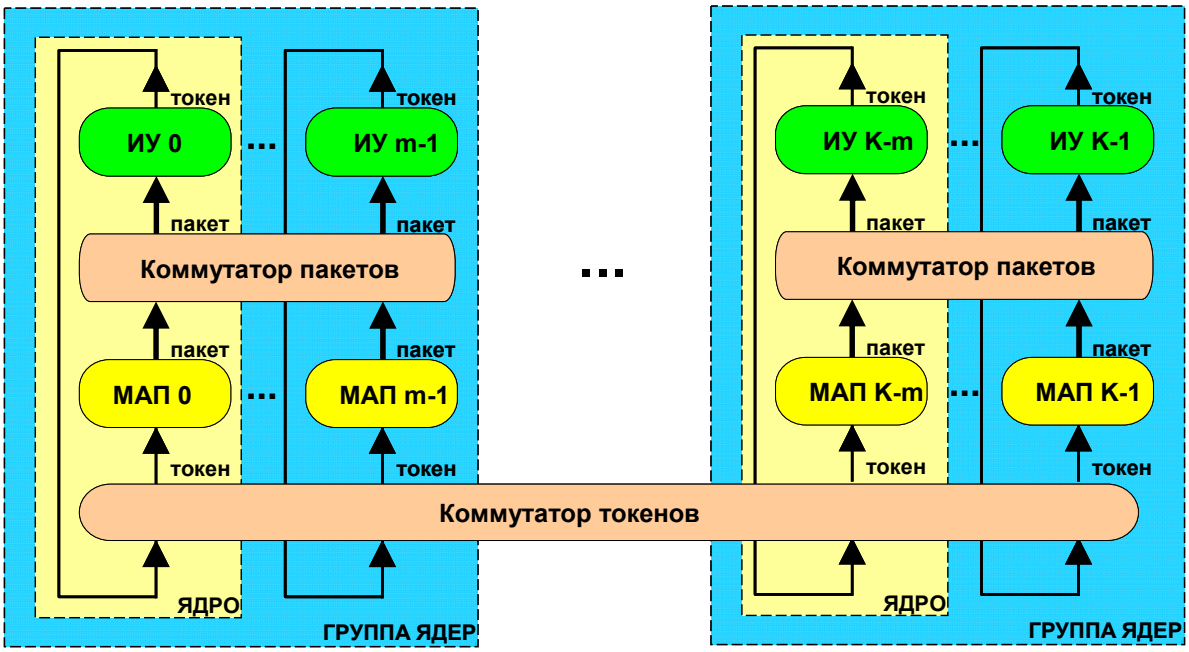

Рис. 1. Структурная схема ППВС

МАП – модуль ассоциативной памяти
ИУ – исполнительное устройство
m – число ядер в группе
K – общее число ядер в системе

ного выполнения своей программы не требуется никаких дополнительных данных; в-третьих, узлы полностью независимы друг от друга, в-четвертых, правильно выборанная хеш-функция (функция распределения вычислений по группам ядер) сокращает число межъядерных передач токенов.

Благодаря такой модели вычислений, реализуемой в архитектуре ППВС, создание программы и ее выполнение не зависят от конкретной конфигурации вычислительной системы, то есть программа будет работать и на одном ядре, и на любом другом числе ядер без повторной компиляции.

ППВС обладает следующими особенностями, актуальными для высокопроизводительных вычислений:

- имеется ассоциативная память с развитой системой команд;
- система хорошо масштабируется, что позволяет создать многоядерный кристалла, а в дальнейшем и высокопроизводительные системы на базе этих кристаллов;
- реализуется аппаратное выявление неявного параллелизма задачи в ходе ее решения;
- имеются аппаратно-программные средства управления параллелизмом задачи;
- имеет место асинхронность работы отдельных блоков системы;
- модель вычислений, реализуемая ППВС, концептуально полностью включает в себя ранее созданные динамическую и статическую модели вычислений с управлением потоком данных;
- потоковая организация вычислительного процесса позволяет нивелировать задержки в коммуникационной сети;
- имеются аппаратно-программные технологии локализации вычислений по пространству и времени;
- для программирования задач используется язык параллельного программирования DFL, позволяющий выразить все преимущества потоковой модели вычислений с динамически формируемым контекстом.

Характерными чертами языка программирования являются:

✓ программисту дана возможность выдавать указания для аппаратных блоков локализации вычислений по вычислительному пространству;

✓ имеется возможность описывать параллельные процессы с учетом недетерминизма их выполнения.

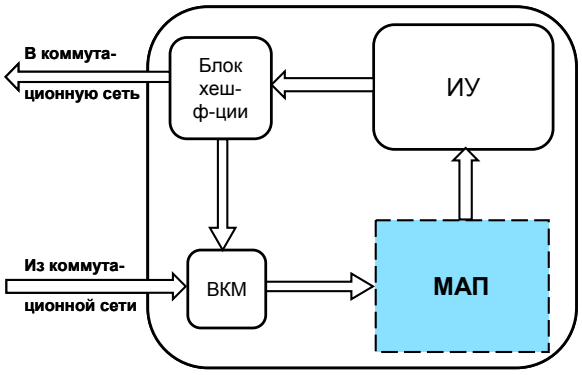

Рис. 2. Блок-схема ядра ППВС

Подробно архитектура параллельной потоковой вычислительной системы рассматривается в работе [1].

Выбор базисных функций распределения – один из путей адаптации ППВС к различным классам задач

Важным инструментом обеспечения высокой эффективности ППВС являются:

а) равномерность нагрузки на вычислительные модули (ядра системы);

б) равномерность нагрузки на коммуникационную сеть.

Эти два показателя часто противоречат друг другу: улучшение по одному показателю приводит к ухудшению по другому. Данные показатели существенно зависят от распределения виртуальных узлов по ядрам ППВС. Поэтому при выборе характера распределения вычислений приходится учитывать влияние распределения на оба показателя.

Таким образом, итоговая производительность системы существенно зависит от выбора способа распределения виртуальных узлов по ядрам для каждой конкретной задачи. Распределение виртуальных узлов осуществляется посредством задания функции распределения (присоединяемой к DFL-программе), аргументом которой является ключ токена, а значением

– номер вычислительного ядра. Эта функция должна вычисляться для каждого посылаемого токена при его создании. Ее значение определяет адрес (или номер) ядра, в которое токен передается посредством коммуникационной сети. Эта функция должна быть «вычислительно дешевой», чтобы не снижалась производительность всей системы. Важно найти хороший базис таких функций, которые могут быть реализованы аппаратно. Выбор такого базиса может осуществляться в рамках настройки и адаптации системы применительно к конкретному классу задач.

Функция распределения [2] в общем случае будет задаваться для каждого узла – тогда она зависит только от контекста и может быть выражена в виде некоторой формулы, параметрами которой являются поля контекста. Основными операциями таких формул могут быть побитовые логические операции, операции сдвига (или умножения, или деление на степень двойки) и некоторые специальные базисные функции. Иногда могут применяться также операции целочисленного сложения или вычитания. Для получения окончательного значения номера ядра берется остаток от деления результата на число ядер K конкретной конфигурации вычислительной системы.

Выбор базисных функций может определяться ориентацией системы на конкретный класс задач. Для таких функций характерно, что они достаточно сложно реализуются программно в традиционной системе команд, что является доводом в пользу их специальной реализации в аппаратуре. Такая реализация возможна либо в форме дополнительных специальных команд, либо в составе отдельного таблично-программируемого блока.

Рассмотрим некоторые примеры таких базисных функций и классы задач, в которых их полезно применять.

Функция zip(i,j) и ее вариации

Основной вариант zip(i,j) принимает два двоичных аргумента и скрещивает их разряды: биты из первого аргумента идут на четные места результирующего аргумента (нумеруем с нуля), второго – на нечетные. Данная функция (и ее вариации) оказывается полезной для задач, где требуются многомерные (2 и более измерений) решетки с доступом к ближайшим соседям, а также для операций над плотно заполненными матрицами.

Функция norm(n,k)

Данная функция «нормализует» первый аргумент n и затем берет k старших разрядов результата. Под нормализацией понимается сдвиг аргумента влево так, чтобы его старшая единица вышла за пределы разрядной сетки числа. Эта функция полезна в алгоритмах свертки по методу сдваивания: вычисление суммы массива, минимума, максимума и подобных ассоциативных операций.

Функция blk(i,N,K)

Здесь N – размер массива, а K – число ядер. Эта функция полезна для распределения элементов одномерного массива длины N с индексами от 0 до N-1. В принципе можно было бы использовать операцию деления, помещая i-й элемент в ядро номер i/B, где B=cell(N/K) – размер блока. Но деление – дорогая операция и потому имеет смысл использовать функцию blk, помещая i-й элемент в ядро с номером blk(i,N,K). Функция blk обеспечивает наилучшую равномерность распределения однородного массива элементов с индексами i от 0 до N-1 на всех уровнях коммуникационной иерархии (в предположении, что иерархическая близость соответствует арифметической, иначе говоря, модули с близкими номерами близки коммуникационно). При $K=2^p$ и произвольном значении N данная функция может быть выражена следующей формулой с использованием функции norm(n,k):

$$blk(i,N,K) = \\ = norm(((i \oplus N) * 2^L + i) * 2 + 1, p) / 2^{L+1-p},$$

где $\oplus$ – операция поразрядного сложения по модулю 2, L – количество разрядов в записи величины N, p – число разрядов в номере ядра. Смысл этой формулы в том, что мы выбираем размер порции в виде степени 2, причем как можно больший, чтобы значительная часть массива была распределена среди всех K ядер порциями такого размера, затем над остатком

проделывается то же и так далее, пока не исчерпаем все элементы.

Функция HMerge(i,j,Li,Lj)

Данная функция может найти применение в алгоритмах с тесной связностью, где требуется комбинировать попарно элементы большого множества, но есть способ сделать это быстрее, чем перебирая N^2 комбинаций. Примером задачи такого рода является алгоритм быстрого преобразования Фурье (БПФ). Предполагая, что размер задачи $N=2^n$ ненамного превосходит число имеющихся процессоров $K=2^p$, требуется его вычислить как можно быстрее, задействовав все процессоры многопроцессорной системы.

Содержательно аргументом функции являются две битовые строки **i** и **j**, которые выражают два индекса двумерного пространства задачи. В ходе алгоритма (по его уровням) первая (**i**) укорачивается, а вторая (**j**) удлиняется. Поскольку функции должны иметь целочисленные аргументы, то добавлены аргументы Li и Lj – длины значений **i** и **j** соответственно.

На содержательном уровне функция HMerge нацелена на то, чтобы ее значение как можно меньше изменялось при переходе от уровня к уровню. Поэтому старшие разряды значения свяжем преимущественно с левыми битами строк (предполагая, что изменения **i** и **j** происходят с правых концов). Поскольку функция «оперирует» строками, проще описать ее работу следующим образом:

1. Примем результат равным пустой строке битов.
2. Если остались строки длин 0 и 1, то переносим оставшийся один разряд **j** в результат. Перейти к п.7.
3. Выбираем более длинную из строк **i**, **j**. Если строки одинаковой длины, то выбираем **i**.
4. Примем **q** (временная переменная) равной половине суммы длин **i** и **j**. Если на шаге 3 была выбрана строка **i**, полусумму возьмем с избытком, иначе – с недостатком.
5. Из выбранной строки отберем q левых разрядов, последовательно перенося их в правый конец результата (выбранная строка станет на q единиц короче).
6. Перейти к п.2.
7. Возьмем p левых разрядов результата.

Эксперименты показали, что при использовании хорошо подобранной функции распределения общее время параллельного выполнения программы значительно сокращается за счет улучшения равномерности распределения и существенного сокращения объема передаваемой информации на всех уровнях иерархии вычислительной системы. На некоторых задачах выигрыш от правильно подобранной функции распределения составляет более чем порядок [1].

Особо стоит отметить, что в вычислительной модели ППВС вопрос распределения решается только применительно к общему вычислительному пространству – пространству виртуальных узлов. Этим ППВС выгодно отличается от систем традиционного параллельного программирования, где приходится отдельно управлять распределением данных и отдельно – распределением вычислений и при этом решать проблему соответствия одного другому во избежание большого количество операций доступа к удаленной памяти.

Другие методы адаптации архитектуры ППВС к различным классам задач

ППВС можно эффективно адаптировать для решения отдельных классов задач иными способами, отличающимися от описанного выше.

Существует, например, возможность, манипулируя количеством ядер (без перепрограммирования задачи) и внутренним наполнением ядра настроить вычислительную систему на решение конкретной задачи. Уменьшение разрядности данных и ключа ведет к уменьшению разрядности функциональных устройств и, таким образом, к уменьшению площади, занимаемой этими устройствами в одном ядре. В итоге позволяет увеличить общее количество ядер на кристалле, а, следовательно, повысить и параллельность обработки данных (поскольку в этом случае одновременно смогут обрабатываться большее количество узлов на независимых ядрах системы) при наличии достаточного параллелизма задачи.

Исключение неиспользуемых команд, в предельном случае, позволяет полностью изба-

виться от ИУ в составе ядра, перенося требуемые функциональные блоки в состав АП и уменьшая тем самым длину вычислительного конвейера. Введение же новых типов токенов и команд в вычислительную систему позволяет для некоторых типов задач существенно оптимизировать их выполнение и в ряде случаев упростить программирование, максимально эффективно используя все преимущества потоковой модели вычислений с динамически формируемым контекстом.

Изменяя топологию коммуникационной среды и в соответствии с ней корректируя функцию распределения (хеш-функцию), локализующую вычисления внутри группы близко расположенных ядер, обеспечивается снижение нагрузки на коммуникационную сеть и уменьшение задержки прохождения данных по «вычислительному кольцу» (АП → ИУ → АП).

Изменяя размеры и алгоритм работы буферов, можно управлять развитием вычислительного процесса (вширь или вглубь) и регулировать тем самым параллелизм вычислений.

Оптимизация выполнения программы на вычислительной системе достигается как адаптацией ее аппаратуры, так и адаптацией самой программы.

Адаптация программы заключается в подстройке ее алгоритма под особенности вычислительной системы, оптимизацию ее выполнения под требования минимального времени выполнения, минимального занимаемого размера ассоциативной памяти, максимального параллелизма и других возможных критериев.

Можно отметить также, что, например, изменение размера программы узла влечет за собой изменение уровня параллелизма [3] и количества токенов, циркулирующих в коммуникационной сети, а выбор оптимальной хеш-функции для локализации сложно организованных данных существенно снижает время прохождения задачи.

Общую методологию разработки вычислительных параллельных систем на основе потоковой модели вычислений с динамически формируемым контекстом можно представить следующим образом.

Выбирается в качестве базового ядра будущей вычислительной системы универсальное ядро ППВС, затем производится адаптация алгоритма задачи под потоковую модель вычислений. На следующем этапе задача программируется на параллельном языке DFL, производится моделирование выполнения задачи на инструментальном комплексе ППВС, на конфигурации, состоящей из базовых ядер и с помощью имеющегося инструментария проводится сбор и анализ статистики (исследуются изменения параметров отдельных узлов ядра и их влияние на прохождение задачи - на время выполнения, на затраты по занимаемой площади и энергопотреблению).

На основе полученных данных на модели выбирается оптимальная по ряду критериев конфигурация ядра и вычислительной системы в целом для этой задачи (или класса задач). Причем к настраиваемым параметрам относятся: емкость АП, размер памяти команд в ИУ, размер регистрового файла, количество функциональных устройств в ядре и их типы, метод управления иерархией памяти, подключение работы с многовходовыми токенами и пакетами, а также другие параметры.

После выбора конфигурации и задания всех параметров формируется новое ядро системы (адаптированное под задачу), выстраивается и моделируется коммуникационная среда, затем проводится моделирование всей системы, выбирается оптимальное количество ядер и составляется техническое задание на разработку кристалла для данного класса задач.

Пример адаптации для задачи БПФ

Алгоритм быстрого преобразования Фурье (БПФ) используется во многих прикладных областях, включая обработку сигналов. Задача БПФ относится к задачам с плохой временной локализацией, и на обычных системах возникают большие задержки при выполнении операций обращения к памяти. Эта задача входит во многие тестовые пакеты для проверки вычислительных систем, поэтому были проведены исследования возможности эффективного выполнения ее на ППВС. Для исследования возможности адаптации базовой архитектуры вычислительной системы под конкретную задачу с целью повышения эффективности ее выполнения была изменена архитектура ядра.

Одним из наиболее распространенных алгоритмов БПФ является алгоритм, построенный по методу Cooley-Tukey (Кули-Таки) [4], который обладает рядом хороших технологических свойств. Структура алгоритма и его базовые операции не зависят от числа отсчетов (меняется только число прогонов базовой операции «бабочка»).

При решении данной задачи на системах с традиционной архитектурой между итерациями алгоритма необходимо выполнять барьерную синхронизацию, что снижает эффективность выполнения задачи. Параллельная потоковая вычислительная система благодаря использованию свойств модели вычислений может одновременно обрабатывать операции «бабочка», принадлежащие разным итерациям алгоритма, что позволяет снизить простои оборудования и повысить эффективность использования аппаратных ресурсов.

«Аппаратная» реализация алгоритма БПФ на ППВС

Все поворачивающие множители вычисляются заранее и хранятся в памяти констант вычислительной системы. В отличие от стандартной (универсальной) структуры ядра (Рис. 2) при аппаратной реализации алгоритма БПФ на поведенческой модели в структуру ядра (Рис.3) и систему команд были внесены следующие изменения:

- в систему команд добавлен новый тип токена – «токен БПФ»;
- в модуль ассоциативной памяти добавлен блок быстрого преобразования Фурье, который выполняет операцию «бабочка», формирует токены для следующих итераций, а также отслеживает момент окончания программы (по анализу одного из полей токена);
- создана хеш-функция (функция распределения), способная уменьшить число пересылок между ядрами вычислительной системы.

Старт выполнения задачи БПФ осуществляется путем посылки токенов с инвертированными индексами и соответствующим кодом операции в токене.

Данная реализация позволяет, как полностью отказаться от исполнительного устройства, так и работать параллельно с ним, являясь

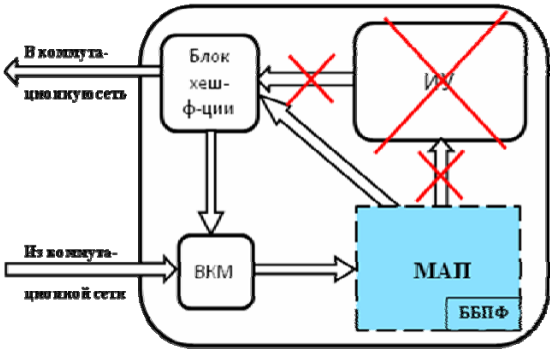

Рис. 3. Структурная схема специализированного ядра (для задачи БПФ)

ББПФ – блок быстрого преобразование Фурье

ускорителем выполнения задачи БПФ, который не использует ресурсы исполнительных устройств системы. Масштабирование осуществляется за счет свойств самой вычислительной системы и модели вычислений.

Данный алгоритм также был реализован на универсальном ядре. Для этого алгоритм, адаптированный под потоковую модель вычислений был запрограммирован на параллельном языке DFL, и была создана функция распределения HMerge (описана в предыдущем разделе), позволяющая хорошо локализовывать вычисления.

Результаты моделирования реализаций алгоритмов

Задачи, реализующие программный и аппаратный алгоритм БПФ, были промоделированы на поведенческой блочно-регистровой модели вычислительной системы [5]. По вертикальной оси рисунков 4-6 отложено число тактов выполнения задачи на поведенческой модели, а по горизонтальной оси – число ядер в конфигурации вычислительной системы (от 16-ти до 64-х) для Рис. 4, 6, и размерность задачи для Рис. 5.

На Рис.4 приведено сравнение времени выполнения аппаратной реализации алгоритма БПФ при использовании различных функций распределения – стандартной функции (16K_STD) и специальной функции распределения HMerge (16K_FFT).

Отчетливо виден эффект применения специальной функции распределения, позволяющей сократить в несколько раз время выполнения задачи за счет уменьшения числа «длинных» передач.

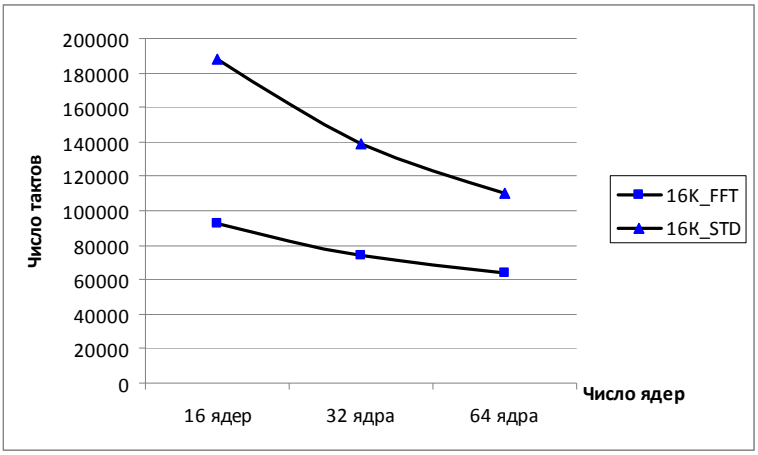

Рис. 4. Сравнение времени выполнения аппаратной реализации алгоритма БПФ при использовании различных функций распределения

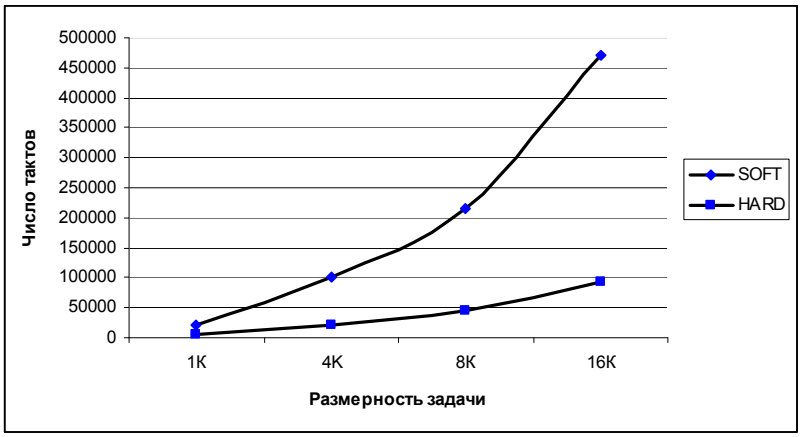

Рис. 5. Сравнение времени выполнения «аппаратной» и «программной» реализаций алгоритма БПФ

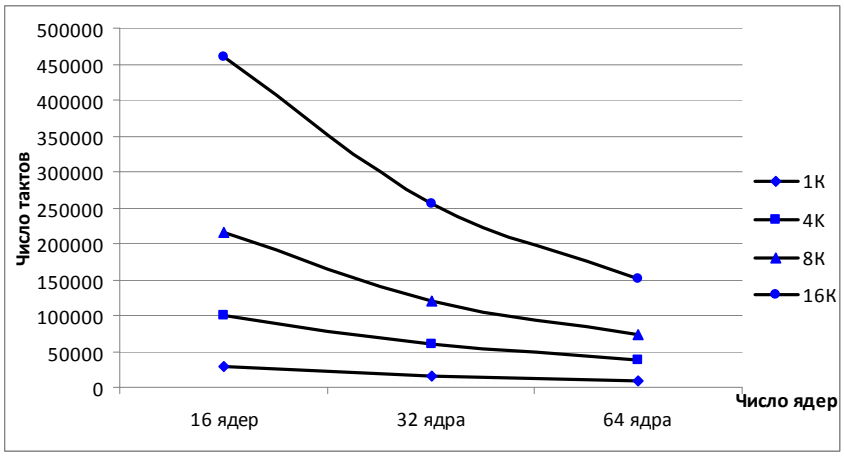

Рис. 6. Сравнение времени выполнения программной реализации алгоритма БПФ различной размерности при изменении числа ядер

На Рис. 5 приведено сравнение времени выполнения «аппаратной» (HARD) и «программной» (SOFT) реализаций алгоритма БПФ различной размерности входного вектора. Результаты моделирования показали, что при небольших размерностях вектора время выполнения программы БПФ для «аппаратной» реализации алгоритма уменьшается в несколько раз, с увеличением размерности входного вектора разрыв между временем выполнения «аппаратной» и «программной» реализаций только растет.

Сравнение времени выполнения «программной» реализации алгоритма БПФ различной размерности при изменении числа ядер представлено на Рис. 6, из анализа которого можно сделать вывод о существенном уменьшении времени выполнения программы БПФ с увеличением числа ядер.

Дальнейшие исследования по применению описанных методов адаптации, в том числе и различных функций распределения, будут продолжены для широкого круга задач с использованием специализированных аппаратных решений на разнообразных конфигурациях параллельной потоковой вычислительной системы.

Заключение

В настоящее время растет необходимость разработки высокопроизводительных вычислительных систем и суперкомпьютеров разной производительности и с разным архитектурным представлением.

В этой связи создается аппаратно-программный комплекс, который позволит получать оптимальную конфигурацию ядра и системы в целом (на основе потоковой модели вычислений с динамически формируемым контекстом) по выбранным пользователем критериям и в зависимости от заданных параметров. Критериями оптимизации могут быть производительность, занимаемая площадь, энергопотребление, размер ассоциативной памяти, количество ядер, требуемая параллельность вычислений и т.п. В качестве параметров данного комплекса могут использоваться максимальный параллелизм задачи, количество входных данных, требуемый темп вычислений и т.п.

Архитектура ППВС масштабируемая, поэтому она может быть изменена для применения ее как при создании суперкомпьютера (что позволит эффективно решать большой круг актуальных задач с высокой реальной производительностью), так и вычислителей специального применения (встроенных систем) с упрощением аппаратной реализации, предназначенных для решения конкретной задачи.

Описанный в статье подход к адаптации вычислительной системы позволяет добиться нового уровня функциональности параллельных высокопроизводительных систем и сократить расходы на их разработку.

Литература

1. Стемпковский А.Л., Левченко Н.Н., Окунев А.С, Цветков В.В. Параллельная потоковая вычислительная система — дальнейшее развитие архитектуры и структурной организации вычислительной системы с автоматическим распределением ресурсов // журнал "Информационные технологии" №10, 2008, С. 2 – 7.
2. Климов А.В., Окунев А.С., Степанов А.М. Распределение вычислений и данных по процессорным модулям вычислительной системы массового параллелизма // Материалы Международной научно-технической конференции «Многопроцессорные вычислительные и управляющие системы» в составе мультиконференции «Проблемы информационно-компьютерных технологий и мехатроники», 2007, т. 1. – Таганрог: Издательство ТТИ ЮФУ, С. 242-246.
3. Левченко Н.Н., Окунев А.С. Аппаратно-программные методы ограничения «взрывного» параллелизма в вычислительной системе с автоматическим распределением ресурсов// Приложение к журналу «Открытое образование», Материалы XXXIII международной конференции «Информационные технологии в науке, образовании, телекоммуникации и бизнесе. (IT + S&E’06)», Гурзуф, 2006. С. 42-43.
4. Блейхут Р. Быстрые алгоритмы цифровой обработки сигналов: Пер. с англ. – М.: Мир, 1989. – 448 с.
5. Левченко Н.Н., Окунев А.С., Змеев Д.Н. Расширение возможностей поведенческой блочно-регистровой модели параллельной потоковой вычислительной системы. // Материалы Пятой Международной молодежной научно-технической конференции «Высокопроизводительные вычислительные системы (ВПВС'2008)», 2008, Таганрог, Россия, стр. 371 – 374.

Климов Аркадий Валентинович. Старший научный сотрудник Института проблем проектирования в микроэлектронике РАН. Окончил Московский государственный университет в 1976 году. Автор более 20 научных работ. Область научных интересов - языки программирования и модели вычислений, семантика языков программирования, методы анализа, преобразования и синтеза программ, нестандартные вычислительные модели и архитектуры, параллельные вычисления и алгоритмы, языки параллельного программирования, архитектуры параллельных ЭВМ, klimov@burcom.ru.

Левченко Николай Николаевич. Старший научный сотрудник Института проблем проектирования в микроэлектронике РАН. Окончил Московский авиационно-технологический институт (технический университет) им. К.Э. Циолковского (МАТИ) в 2001 году. Кандидат технических наук. Автор более 40 научных работ, включая монографию и патент на изобретение. Область интересов - параллельные вычисления, параллельное программирование, модель вычислений управляемых потоком данных, использование ассоциативной памяти в высокопроизводительных вычислительных системах, nick@ippm.ru.

Окунев Анатолий Семенович. Старший научный сотрудник Института проблем проектирования в микроэлектронике РАН. Окончил Московское высшее техническое училище им. Н.Э. Баумана (МВТУ) в 1974 году. Кандидат технических наук. Автор более 40 научных работ, включая монографию и патент изобретение. Область интересов – архитектуры высокопроизводительных вычислительных систем, модель вычислений с управлением потоком данных, применение различных САПР для реализации архитектурных решений, применение принципов ассоциативности при разработке различных аппаратных узлов, oku@ippm.ru.

Стемпковский Александр Леонидович. Директор Института проблем проектирования в микроэлектронике РАН. Окончил Московский государственный институт электронной техники (технический университет) (МИЭТ) в 1973 году. Доктор технических наук, профессор, академик РАН. Автор 120 научных работ, включая три монографии и изобретения. Область научных интересов - фундаментальные проблемы построения систем автоматизации проектирования СБИС, методология проектирования сложных микроэлектронных устройств, ippm@ippm.ru.