# ПРОБЛЕМЫ РЕАЛИЗАЦИИ МАССОВОГО ДИНАМИЧЕСКОГО ПАРАЛЛЕЛИЗМА. I

А.В. Махиборода, А.В. Ильичёв, А.А. Подобин

*Департамент прикладной математики МИЭМ,*

*Национальный исследовательский университет «Высшая школа экономики»*

makhiboroda@yandex.ru

Поступила 01.11.2015

Современное состояние технологии микроэлектронного производства позволяет разместить десятки тысяч процессорных элементов на одном кристалле и объявлены тенденции к устойчивому росту этого показателя на ближайшее десятилетие. При этом рост тактовой частоты остановлен и зафиксирован на оптимальном значении 2.2 ГГц. Освоение массового параллелизма становится основным инструментом обеспечения роста производительности вычислительных средств. Однако на пути реализации массового параллелизма вычислений существует целый ряд препятствий, требующих решения принципиальных проблем. В статье рассмотрен ряд примеров, иллюстрирующих эффект быстрой остановки роста производительности при наращивании значений параллелизма. Проведен анализ причин подавляющих рост производительности на начальных шагах роста параллелизма. Предлагаются меры по переносу точки насыщения роста в область более высоких значений числа процессорных элементов. Обсуждаются направления развития архитектуры и принципы организации вычислений, имеющие перспективу эффективной реализации массового динамического параллелизма.

УДК 681.3.06

## Введение

Компьютерное моделирование и постановка численных экспериментов являются важнейшими инструментами исследования наноструктурных объектов в различных отраслях научного знания и в наукоёмких технологических разработках. Не пытаясь описать огромную литературу по этой тематике, мы сошлемся только на обзоры, опубликованные в журнале «Наноструктуры. Математическая физика и моделирование» по компьютерному моделированию в биологии [1], по проблемам параллельного программирования [2], по проблемам создания и функционирования баз данных [3] и другим направлениям развития и применения вычислительных средств [4, 5, 6].

В данной статье делается попытка анализа глубокого перелома в эволюции компьютерных систем, который медленно вызревал не одно десятилетие и сейчас проявился с полной определённостью. Сегодня можно утверждать, что главным направлением развития вычислительных средств на ближайшую перспективу станет освоение ***массового динамического параллелизма***.

Массовый параллелизм это не 5, не 10 и даже не 100 процессоров, а сотни тысяч и миллионы процессорных элементов, работающих одновременно, согласованно и взаимосвязанно. В этих условиях синхронный подход к программированию, предполагающий опережающую разметку трассы процесса по каждой ветви становится бессмысленным и неосуществимым. Массовый параллелизм может быть только динамическим, то есть возникающим оперативно шаг за шагом в ходе текущего осуществления вычислительных операций. При реализации динамического параллелизма вычислительный процесс должен сам прокладывать свой путь в аппаратной среде и быть способным эффективно заполнять высокопараллельный аппаратный ресурс.

Ситуация, открывающая принципиально новый этап развития вычислительных средств определяется текущим состоянием технологии микроэлектронного производства. Выделим три принципиальные позиции, характеризующие современное состояние технологии.

Позиция первая: рост тактовой частоты остановлен и зафиксирован на уровне 2,2 ГГц. В предыдущие годы были достигнуты значения тактовой частоты на уровне 5,5 ГГц, но при этом получен неприемлемый рост энергозатрат. Технологи приняли решение вернуться к оптимальному значению 2,2 ГГц и более тактовую частоту не наращивать. Это означало необходимость пересмотра «закона Мура», в соответствии с которым технологи обязаны обеспечивать удвоение производительности кристалла каждые 18 месяцев.

Прогнозируемый рост числа транзисторов на кристалле носит линейный характер и пока не встречает ограничений. Отсюда следует вторая принципиальная позиция: устойчивый рост числа транзисторов позволяет переформатировать «закона Мура» и декларировать удвоение числа ядер на кристалле через каждые 18 месяцев (или каждые 24 месяца). На текущий момент в производстве достигнуты значения 2688 ядер на одном кристалле, выпускаемом фирмой NVIDIA [7] и прогнозируется их дальнейший рост. Это и есть факт технологического оформления коренного перелома в развитии вычислительных средств. Освоение массового параллелизма становится основным инструментом обеспечения устойчивого роста производительности кристаллов. Отсюда неумолимо следует необходимость обновления технологии программирования. Неоднократно разными специалистами высказывалось мнение о том, что «теперь программирование придётся изобретать заново» [8]. Это очень показательное заявление, обозначающее смену вектора развития. Предшествующий этап развития вычислительных средств характеризовался ограничениями разработок в области архитектуры и организации вычислений. Проблема состояла в том, что в мире был наработан огромный задел программных продуктов, оцениваемый в сотни миллиардов долларов. Основным условием развития вычислительных средств было обеспечение преемственности и прямой переносимости программных продуктов и технологий, наработанных ранее. Теперь давление этого фактора заметно снижается, компьютерное сообщество объявляет о готовности к пересмотру и обновлению архитектурных концепций.

И, наконец, третья позиция. При повышении тактовой частоты расход энергии растёт как квадрат частоты. Передовые массивно-параллельные системы потребляют мегаватты энергии. Стоимость энергии начинает превышать стоимость оборудования. Энергопотребление ограничивает будущий рост мультипроцессорных систем. Поэтому актуально введение нового понятия вычислительной эффективности - performance/watt вместо пиковой производительности. Этим показателем активно пользуется фирма Intel

при описании своей продукции. В упрощённом виде это величина отношения числа операций на 1 ватт потребляемой энергии. Специалисты фирмы SUN предложили ввести более сложный критерий энергоэффективности SWaP (Space, Watts and Performance). Эта тема активно обсуждается, например, в [9].

Введение критерия энергоэффективности кристалла косвенно связано с числом транзисторов и означает явно сформулированное требование эффективного использования основного ресурса кристалла, который и определяется числом транзисторов. Теперь разработчики архитектуры должны ответить на вопрос – каково соотношение затрат транзисторов на обработку данных в арифметико-логическом блоке и на управление в других блоках процессора. Показатель степени интеграции в современных технологиях достиг порядка миллиарда транзисторов на кристалле и далее будет расти [10]. Однако каким бы безбрежным не казалось это море транзисторов и сколь бы ничтожна мала не оказалась их стоимость, теперь разработчики архитектуры обязаны отвечать за показатель энергоэффективности кристалла, а это требует рационально распределять транзисторный ресурс при проектировании.

Перечислим ещё раз три основополагающие позиции, фиксирующие современное состояние технологии микроэлектронного производства:

- Тактовая частота зафиксирована на уровне 2.2 ГГц и её дальнейший рост считается нецелесообразным;
- Закон Мура предписывает удвоение числа ядер на кристалле каждые 18 (или 24) месяцев;
- Вводится критерий энергоэффективности кристалла, заданный как число операций на единицу затрат энергии.

Таким образом, можно констатировать, что сложились условия, критерии и объективные оценки, стимулирующие проведение фундаментальных разработок в области архитектуры перспективных вычислительных средств, которые должны решить следующие три проблемы:

- обеспечить реализацию массового параллелизма;
- обеспечить высокую эффективность организации вычислительных процессов по критерию числа операций, приведенных к энергозатратам:
- создать предпосылки для разработки принципиально новых технологий программирования.

## 1. Некоторые этапы из истории освоения массового параллелизма

История освоения параллелизма полна драматических разочарований и непрогнозируемых проявлений побочных и скрытых эффектов. Первичные ожидания устойчивого монотонного роста производительности в широком диапазоне значений числа обрабатывающих элементов в общем случае не подтвердились. Определённые положительные результаты получены только в случаях узкой специализации параллельных структур на определённый вид параллелизма, на определённый класс задач или на одну конкретную задачу.

С самого начала было понятно, что возможности распараллеливания вычислений в первую очередь ограничиваются свойствами решаемой задачи. Если в задаче выделяется 10 параллельных ветвей, наращивание числа обрабатывающих процессоров свыше десяти становится бессмысленным. График роста производительности системы в ре-

жиме решения данной задачи достигнет своего максимума при 10 процессорах и деле расти не будет. Более полно факт зависимости роста производительности от свойств задачи сформулировал Амдал ещё в 1967 году [11]. В законе Амдала вводится основная характеристика параллельной системы в виде коэффициента ускорения $S_p$ , который определяется как отношение времени выполнения задачи на одном процессоре $T_1$ ко времени выполнения этой задачи на параллельной системе из $P$ процессоров $T_p$. Поскольку $S_p$ величина относительная, можно считать, что $T_1$ в числителе равно единице. Для определения $T_p$ временной баланс разбивается на две составляющие – последовательную, которая не поддаётся распараллеливанию и обозначается как доля от общего объёма вычислений $\alpha$, и параллельную, обозначаемую как $1-\alpha$. Доля $1-\alpha$ поддаётся распараллеливанию и с ростом числа процессоров уменьшается, а доля $\alpha$ остаётся неизменной. Тогда ускорение $S_p$ может быть записано как соотношение:

$$S_p = \frac{1}{\alpha + \frac{1-\alpha}{p}}$$

Ниже на рис. 1 приводятся графики роста ускорения $S_p$ для разных вариантов соотношения параллельных и последовательных долей в задаче.

Закон Амдала наглядно иллюстрирует важный факт, говорящий о том, что каждая конкретная задача имеет определённый потенциал параллелизма и при исчерпании этого потенциала значение ускорения $S_p$ достигает своего предельного значения и далее не растёт при любом количестве процессорных элементов.

Закон Амдала даёт идеальную картину, в которой предполагается, что параллельная компонента задачи может дробиться и поглощаться параллельной структурой обработки неограниченно. В реальных условиях определение соотношения долей параллельных и последовательных компонент задачи, а также уровень дробления параллельной компоненты при распараллеливании зависят от архитектуры вычислительной среды, от методов программирования, размерности задачи и масштабов представления базовых фрагментов (так называемое мелкозернистое и крупнозернистое представление). В реальных условиях насыщение роста ускорения $S_p$ наступает раньше и при меньших значениях.

Движение в сторону максимального использования массового параллелизма предписывает выбирать крупноразмерные задачи и стремиться к их мелкозернистому представлению. А это в свою очередь создаёт значительные проблемы в области программирования, выбора архитектуры аппаратных средств и форм организации вычислений. Первые разработки высокопараллельных структур предпринимались в рамках классической фон Неймановской архитектуры, которая изначально предназначалась для построения сугубо последовательных синхронных форм функционирования. Попытка построения параллельной структуры как совокупности многочисленных последовательных синхронных ветвей, функционирующих параллельно, представляет собой сложную задачу с многочисленными проблемами и скрытыми побочными эффектами.

Значительным достижением в области построения параллельных систем была разработка в MIT архитектуры потока данных, известной как архитектура Data Flow. Первые публикации по архитектуре Data Flow относятся к середине 70-х [12]. Формы организации вычислений в архитектуре Data Flow в корне отличаются от принятых в классической фон Неймановской машине и изначально ориентированы на параллельное функционирование. Архитектура Data Flow функционирует асинхронно и самостоятельно извлекает динамический параллелизм по текущему состоянию и готовности данных.

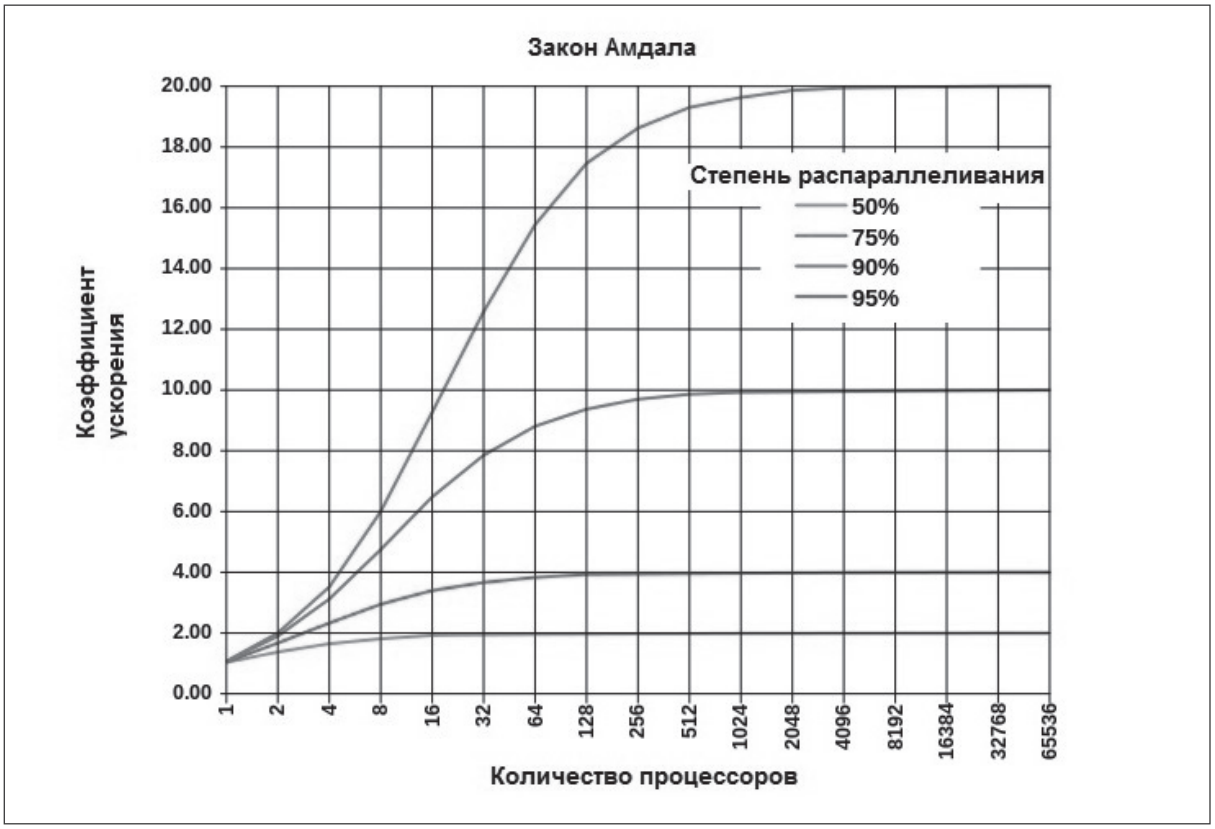


Рис. 1. Графики роста ускорения $S_p$ для разных вариантов соотношения параллельных и последовательных долей в задаче.

Язык машины потока данных содержит базовые лексические конструкции двух типов – пакет и фишка. Пакет представляет собой составную запись определённого формата, состоящую из определённого набора именных полей, предназначенную для описания элементарной процедуры вычислительного процесса. Структура записи пакета приведена на рис. 2.

Пакет описывает элементарную процедуру преобразования двух аргументов. Именные поля пакета обозначают имена входных аргументов, имя функции и имя результата. В отличие от классической машины базовые операции и форматы данных, поименованные в полях пакета, не фиксированы конструкцией машины, а создаются при программировании конкретной задачи. Кроме именных полей в пакете есть поле спусковых флагов. Флаг - это один бит, а поле флагов - битовый вектор. Часть флагов ассоциированы с именами аргументов, поименованных в пакете. Флаги могут быть взведены либо опущены. На флагах реализуются спусковые функции, которые определяют статус пакета по заданным критериям и могут осуществлять перевод пакета из пассивного состояния ожидания в состояние готовности к исполнению. Совокупность пакетов описывает вычислительный процесс в виде графа, поскольку данные, поименованные в полях одних пакетов как результаты, поименованы в других пакетах как аргументы. С точки зрения графового представления пакет есть форма описания вершины графа. Назовём структуру, представленную совокупностью пакетов, потоковым графом. Пакеты размещаются в специальной памяти хранения программы, порядок размещения пакетов в памяти произвольный, при этом связность потокового графа поддерживается взаимосвязью имён данных в именных полях пакетов.

Данные размещаются в отдельной памяти данных. В памяти данных поддерживается доступ к данным по их именному обозначению. Кроме того память данных ведёт ка-

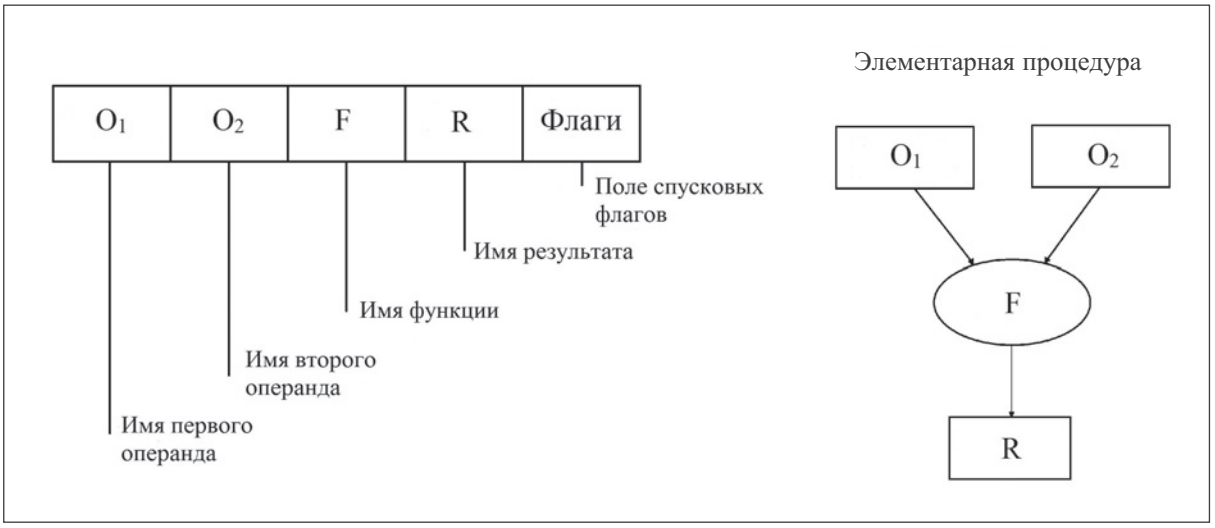


Рис. 2. Структура записи пакета

талог наличия данных и по мере их поступления или удаления вырабатывает фишки с именами соответствующих данных. Фишки направляются в память программ, где по именам, записанным в фишках, осуществляется ассоциативный опрос именных полей пакетов. Если опрос даёт положительный отклик, в поле флагов пакета происходят преобразования. Ассоциированный с данным именем флаг либо взводится в активное состояние, либо опускается. Таким образом, динамика движения данных и их текущее состояние отображаются на состояниях флагов пакетов в памяти программ. Флаги всех пакетов обрабатываются по логике спусковых функций, при этом главным условием спуска является наличие данных, поименованных в пакете в качестве аргументов. В случае готовности аргументов пакет переводится в активное состояние и выставляет маркер готовности. (Два флага, ассоциированные с двумя аргументами пакета поддерживают основной приём формирования спусковых функций по готовности данных кроме того в поле флагов могут быть и дополнительные флаги, расширяющие возможности управления активностью пакетов). Таким образом, в каждый текущий момент времени в памяти программ может возникать множество активных пакетов, которые одновременно передаются в исполнительную подсистему для реализации элементарного кванта вычислений, заданного пакетом. Из памяти программ в исполнительную подсистему транспортируется множество пакетов, которые формируют загрузку параллельной многоядерной среды исполнительной подсистемы. Система Data Flow функционирует лавинообразно – поступающий в систему поток данных отображается на состоянии спусковых флагов и инициирует спуск на обработку множества пакетов, в результате обработки пакетов образуются новые данные, которые заносятся в память данных и спускают на обработку новые порции пакетов и так далее до прекращения поступления данных в систему.

В архитектуре потока данных программируются не последовательности процедур, а взаимосвязи данных и условия спуска пакетов. Вычислительный процесс развивается асинхронно, а логика взаимодействия данных и спуска пакетов самостоятельно извлекает текущий динамический параллелизм из реального состояния данных. Такой подход к организации вычисление имеет предпочтительные перспективы в условиях массового параллелизма. Однако практическая реализация архитектуры Data Flow оказалась проблематичной и факты таковы, что с 70-х годов и до наших дней не существует ни одной промышленной версии машины потока данных. С точки зрения организации

параллельных вычислений это значительный прогресс в сравнении с последовательной синхронной динамикой функционирования классической машины. Но при этом теряется простота и ясность программирования, сводящаяся к формированию последовательных символьных строк в классической машине. В строчку можно писать сотни и тысячи операторов и затем читать их и править, а нарисовать потоковый граф из сотен и тысяч вершин и искать на нём ошибки весьма проблематично.

Параллельно и независимо от проекта Data Flow и одновременно с ним велись работы по непроцедурным стилям программирования и в частности по функциональным языкам программирования [13]. Работы по функциональным языкам привели к необходимости разработки специальных машин, поддерживающих функциональные языки. Один из проектов завершился созданием макетного образца продукционной машины ELIZE [14]. Средствами функционального языка создаётся формализм, называемый продукционной системой. В общем виде абстрактная продукционная формальная система состоит из набора базовых символов и списка продукций. Продукции - это предписания по подстановкам или заменам. Если мы ставим своей целью построение некоторой порождающей продукционной системы, то продукции должны быть расширяющими, т.е. в левой части продукции должен помещаться один исходный символ, а в правой части продукции должен содержаться набор символов, заменяющих исходный. Схема продукции выглядит, например, следующим образом:

$$R : S_{\text{исх}} \rightarrow A, F, K$$

Запись продукции содержательно означает: продукция R предписывает исходный символ $S_{\text{исх}}$ , записанный в левой части продукции заменить на символы $A$, $F$, $K$, записанные в правой части продукции.

Далее начинает действовать редукционный процессор. На вход редукционного процессора подаётся исходная запись в виде некоторой цепочки начальных символов. Редукционный процессор читает исходные символы по одному и совершает поисковые операции в списке продукций. При нахождении продукции, содержащей заданный символ в левой части продукции происходит чтение соответствующей правой части и замена исходного символа на символьный набор, в соответствии с предписанием продукции. Запуск редукционного процесса приводит к пошаговому росту символьной записи. Если мы хотим, чтобы рост был сходящимся, необходимо дифференцировать набор базовых символов и разбить их на терминальные $T$ и нетерминальные $N$. Набор терминальных символов $T$ обозначает некие осмысленные базовые компоненты, из которых должны состоять символьные последовательности, являющиеся продуктами порождения данной продукционной системы. Нетерминальные символы $N$ обслуживают процесс порождения и, как правило, обозначают наборы подмножества и классы базовых компонентов $T$. Набор продукций строится таким образом, что в левой части всех продукций содержатся только нетерминальные символы $N$. Другими словами продукции к терминальным символам не применяются. В правой части продукции могут быть любые символы из $T$ или $N$. Продукционный процесс прекращается, когда все символы порождаемой записи состоят только из терминальных символов.

Идея построения функциональных языков предполагает создание продукционной знаковой системы, позволяющей создать исходную запись задачи как некую содержательную математическую постановку и далее запустить редукционный процесс, который переработает исходную содержательную запись задачи в символьную конструкцию, которая может быть интерпретирована как процедура её решения. Функциональные языки программируют целые классы задач, поскольку в рамках продукционной си-

стемы имеются множества траекторий развития редукционного процесса и, следовательно, порождаются целые классы символьных конструкций. Класс задач поддерживается выбором элементарных операций, образующих функционально полный базис по данному классу. Таким образом, создаётся базовый набор программ и структур данных, обозначаемых как атомарные символы. Атомарные символы не подлежат редуцированию. Это и есть терминальные символы продукционной системы, они выводятся из редукционного процесса и дают начало процессу решения задачи.

Экспериментальная версия редукционной машины ELIZE представляет собой открытый набор стандартных фон Неймановских процессорных элементов с локальной памятью и объединённых общей шиной. Физическая реализация осуществлена на одноплатных транспьютерах промышленного производства. (Сведения о транспьютерах можно найти, например, в [15]. Транспьютер это всего лишь классический процессор с локальной памятью и четырьмя портами для связи со смежными процессорами. По замыслу разработчиков на транспьютерных элементах предполагалось строить масштабируемые матричные структуры). При построении макетного образца редукционной машины применялся метод эмуляции, при котором требуемый набор нестандартных функциональных элементов создаётся путём программирования стандартных классических процессоров. Набор процессорных элементов позволяет распараллелить редукционный процесс, а по мере появления нередуцируемых символов, обозначающих атомарные функции, запускается процесс их исполнения на тех же процессорах.

В функциональных языках постулируются определённые правила конструирования функциональных записей, которые обозначают функциональный оператор, аргументы и результаты. Символы в функциональной записи различаются как редуцируемые и атомарные. Атомарные символы это ссылки на базисные функции, инициирующие вычислительный процесс, а редуцируемые символы поддерживают редукционный процесс, в ходе которого исходная запись разрастается и превращается в запись процесса решения задачи средствами базисного набора. Мы намеренно обошли семантические проблемы построения исчислений, лежащих в основе функциональных языков. Для целей анализа проблем реализации массового параллелизма нам необходимы лишь общее представление о механике функционирования редукционной машины, интерпретирующей функциональные языки и не более того.

Структура функциональных записей в функциональных языках подобна структуре пакета в архитектуре Data Flow. Самое поверхностное знакомство с механикой работы машины потока данных и редукционной машины наталкивает на мысль о том, что проблема программирования архитектуры потока данных может быть решена средствами редукционной машины, интерпретирующей функциональные языки. При обеспечении совместимости правил построения функциональных записей и стандартов построения Data Flow пакетов результаты работы редукционной машины в части реализации редукционного процесса можно воспринимать как генерацию потокового графа для машины потока данных. Именно эта мысль получила серьёзное обоснование в известном докладе Бэкуса. В 1977 году был опубликован доклад Бэкуса, прочитанный им по поводу вручения премии Тьюринга [16]. Доклад называется – «Можно ли освободить программирование от стиля фон Неймана? Функциональный стиль и соответствующая алгебра программ». Доклад подводил итог под предшествующей дискуссией, заполнявшей периодику 70-х годов и известной как кризис архитектуры. Доклад содержит глубокий критический анализ фон Неймановской архитектуры и убедительно показывает исчерпание её возможностей. Во второй части доклада излагается программа развития вычислительных средств на ближайшее будущее, в основу которой положен тезис об объединении двух направлений разработок – проектов Data Flow и функциональных языков. И, наконец, третья часть доклада посвящена изложению основ функ-

ционального языка. В соответствии с программой Бэкуса работы по перспективным технологиям программирования должны быть сосредоточены вокруг разработки функциональных языков, а в части построения аппаратных средств с массовым параллелизмом необходимо принять за основу проект Data Flow.

## 2. Основные характеристики проекта редукционно-потоковой машины

Несмотря на популярность доклада и высокие индексы его цитирования разработчики не спешили откликнуться на призывы Бэкуса и объединить свои усилия. Традиция раздельного существования аппаратных и программных разработок оказалась сильнее здравого смысла. Мы опишем единственный известный нам проект предпринятый автором данной статьи в соответствии с программой Бэкуса в конце 80-х годов. Проект назывался редукционно-потоковая машина, основные сведения по проекту содержатся в патенте [17].

Структурная схема редукционно-потоковой машины приводится на рис. 3.

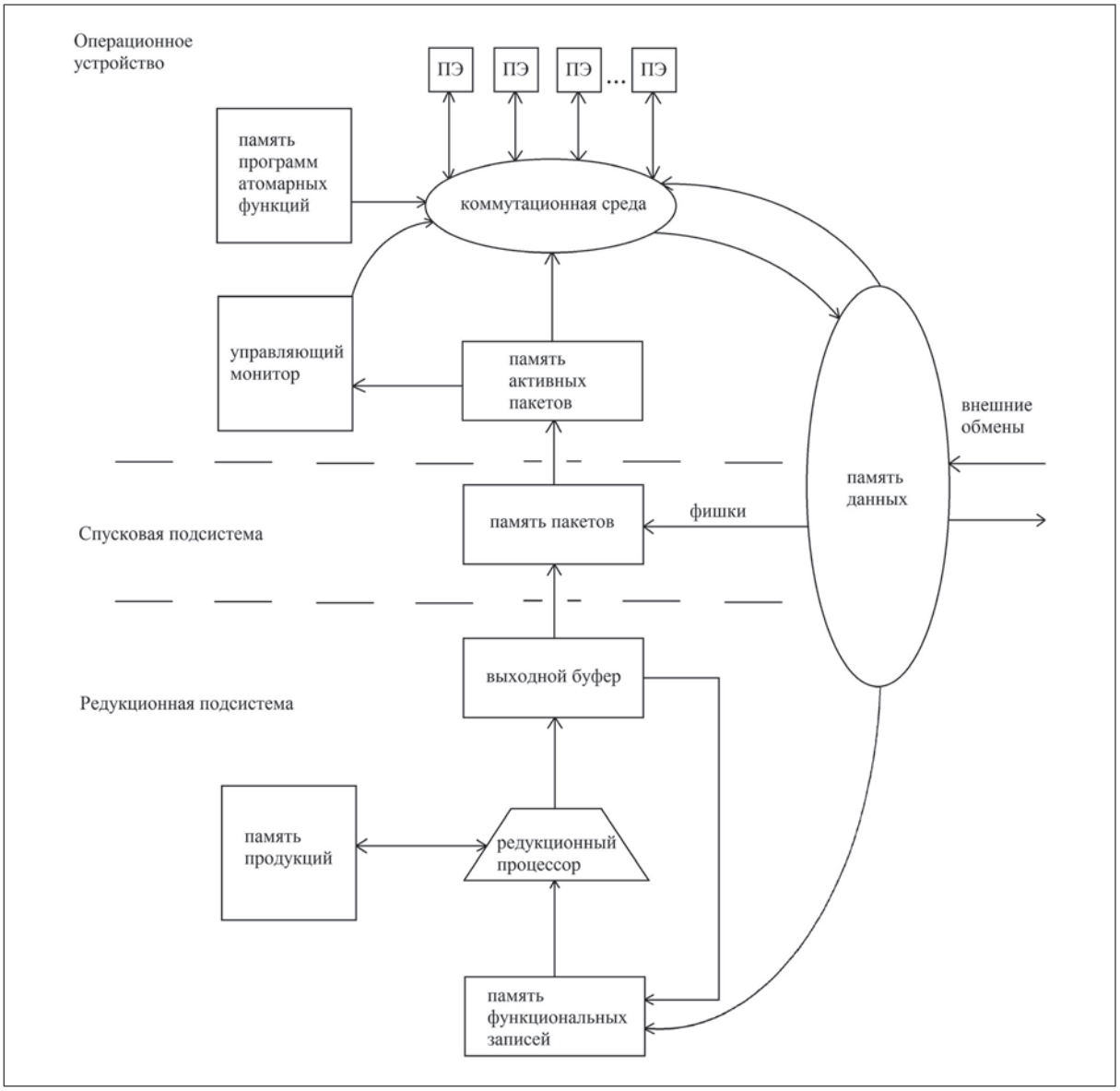


Рис. 3. Структурная схема редукционно-потоковой машины

Машина состоит из трёх подсистем, имеющих доступ к общей памяти данных. Кроме того, каждая подсистема имеет свою локальную функционально ориентированную память, поддерживающую её специфические функции.

Редукционная подсистема представляет собой определённый вариант редукционной машины, интерпретирующей функциональный язык и состоит из редукционного процессора, памяти функциональных записей и памяти продукций. В память функциональных записей заносится исходная цепочка символов и запускается редукционный процесс. Исходной единицей функциональных записей является пакет. Структура пакета уже рассматривалась и была представлена на рис. 2.

Пакет - это битовая строка, состоящая из нескольких полей, фиксирующих символьное обозначение двух аргументов, функции и результата. Пакет представляет элементарную процедуру процесса. Символы, размещённые в полях пакета, могут быть редуцируемыми и служить агентами редукционного процесса, который осуществляет редукционный процессор. Процессор обращается в память продукций и выбирает продукцию соответствующую редуцируемому символу и далее осуществляет замену исходного символа на символьную цепь, предписанную продукцией. В ходе развития редукционного процесса исходная запись разрастается, пополняется новыми символами до тех пор, пока не появятся нередуцируемые символы, представляющие атомарные функции, которые могут быть исполнены в данной продукционной системе. Атомарные (исполняемые) функции оформляются в записи как пакеты и выводятся из редукционной подсистемы. На выходе редукционной подсистемы. формируется поток исполняемых пакетов, который направляется на вход спусковой подсистемы.

Спусковая подсистема это специализированный сокращённый вариант машины потока данных. Основу составляет память хранения пакетов и выполнения спусковых функций. Покидающие редукционную подсистему пакеты оформляются как Data Flow пакеты и кроме полей для наименования функций, аргументов и результатов содержат поле спусковых флагов, предназначенных для выполнения спусковых функций. Множество пакетов, заполняющих спусковую подсистему, в совокупности представляет запись графа вычислительного процесса, поскольку именные поля пакетов взаимосвязаны. Результаты одних пакетов служат аргументами для других пакетов. Таким образом, потоковый граф это предварительная разметка трассы вычислительного процесса. Память данных является многофункциональной и кроме собственно хранения данных осуществляет ряд дополнительных функций, в частности ведёт каталог наличия данных и посылает в спусковую подсистему сообщения о появлении и удалении данных. Сообщения реализуются путём посылки в спусковую подсистему фишек с именами данных. Фишки взаимодействуют с именными полями пакетов по цепям ассоциативного доступа и управляют состояниями спусковых флагов. Если в памяти данных появляются данные с определённым именем, в спусковую подсистему направляется фишка с этим именем. Фишка опрашивает одновременно все именные поля пакетов и в случае совпадения имён взводит соответствующий флаг в поле спусковых флагов всех откликнувшихся пакетов. Если данные выводятся из памяти, посылка фишки должна приводить к опусканию соответствующих флагов в записях пакетов. Таким образом, состояние памяти данных отображается на состояния спусковых флагов пакетов. Далее работает логика спусковых функций по принципу готовности пакета к исполнению по критерию готовности аргументов в памяти данных. Если оба спусковых флага взведены, пакет переводится в активное состояние и выставляет маркер готовности. Память пакетов устроена таким образом, что все пакеты, помеченные маркером готовности, выталкиваются в выходные буфера и далее направляются в операционное устройство. Таким образом, система извлекает текущий параллелизм динамически по текущему состоянию данных.

В данной сокращённой версии машины потока данных принято правило, по которому пакеты, выведенные из памяти для исполнения, в памяти пакетов уничтожаются, т. е. пакеты используются однократно. Это значительно упрощает конструкцию потоковой машины и избавляет от необходимости решения проблемы многократного использования стереотипных фрагментов потокового графа. Для повторного исполнения стереотипных процедур многократно запускается процесс порождения требуемого фрагмента в редукционной подсистеме. Кроме того, в редукционной подсистеме решается проблема связывания повторных реализаций процедуры с другими экземплярами однотипных данных через расстановку индексных меток в полях, именующих аргументы.

На выходах спусковой подсистемы создаётся поток исполняемых пакетов, которые ссылаются на исполняемые функции и гарантированы готовностью данных, на которые ссылаются поля, именующие аргументы. Исполняемые пакеты перемещаются в операционное устройство. В состав операционного устройства входит набор простейших процессорных элементов с небольшой локальной памятью, коммутационная среда для эффективной загрузки и выгрузки данных, память хранения программ атомарных функций, а так же управляющий монитор с локальной памятью для приёма потока исполняемых пакетов. Операционное устройство связано с памятью данных раздельными каналами доступа по записи и чтению. Управляющий монитор принимает исполняемые пакеты и организует загрузку процессорных элементов по содержанию именных полей пакетов. В первоначальной версии проекта предполагалось иметь в составе исполнительной подсистемы 16 – 32 параллельных каналов обработки. Описанная структура рассматривалась как базовый системный кластер, составляющий основу для масштабирования и построения более крупных систем обработки данных.

## 3. Результаты моделирования редукционно-потоковой системы, эффект насыщения роста производительности.

Изложенный выше эскизный проект редукционно-потоковой машины можно воспринимать как определённый этап концептуальной проработки базовых конструкций в соответствии с программой Бэкуса. Действительно в данном проекте просматривается перспектива освоения массового параллелизма и существенного обновления технологии программирования. Естественным шагом продвижения проекта явилась разработка имитационных моделей и постановка модельных экспериментов. К этому времени проект прошёл патентную экспертизу и получил положительные решения по ряду заявок. Ни у кого не возникало сомнений, в работоспособности проекта в целом. Целью постановки модельных экспериментов было уточнение параметров обменных потоков и необходимых объёмов буферных зон на стыках подсистем. Главной проблемой было обоснование необходимых объёмов памяти хранения пакетов в спусковой подсистеме, поскольку это самый дорогостоящий элемент, содержащий обширные поля ассоциативного доступа. В тоже время память пакетов является критическим звеном системы, в котором возможна ситуация зависания процесса вследствие несогласованности темпов порождения пакетов редукционной подсистемой и темпов спуска пакетов по готовности данных.

Программа разработки моделей и постановки модельных экспериментов была успешно выполнена. Структура моделей и основные результаты моделирования изложены в [18]. Однако кроме перечисленных целей моделирования дополнительно была проведена модельная оценка производительности системы. С этой целью были построены кривые роста коэффициента ускорения $K$, который представляет собой отношение времени выполнения задачи на одном процессоре к времени выполнения задачи в многопроцессорной системе. При этом временные затраты оценивались в числе квантов мо-

дельного времени. По результатам модельных экспериментов был построен график зависимости коэффициента $K$ от числа каналов обработки в модели. Число каналов обработки равноценно числу процессорных элементов в исполнительной подсистеме. График приводится на рис.4. На графике обозначено несколько кривых для разных значений средней трудоёмкости пакетов $t_n$ , выраженной в числе квантов модельного времени.

Модельный эксперимент позволил выявить достаточно жёсткую зависимость коэффициента ускорения $K$ от трудоёмкости пакета. Это проявление конкретных значений соотношения временных затрат на обработку и на обменные пересылки данных и загрузку пакета. Если задача разбита на множество мелких пакетов доля затрат времени на обменные пересылки данных оказывается более высокой чем для пакетов с большой трудоёмкостью. При мелких пакетах на рис. 4 график демонстрирует нечувствительность к параллелизму. Эффект роста производительности при распараллеливании задачи начинает проявляться по мере роста трудоёмкости пакетов. Это неожиданный удар по идее мелкозернистого представления задачи с целью более эффективного использования возможностей параллелизма. Возникает ограничитель понижения масштаба мелкозернистого представления задачи.

Однако главное заключается в том, что реальные значения коэффициента ускорения $K$ оказываются неприемлемо низкими, а график роста входит в насыщение при относительно малом значении параллелизма – при 6 каналах обработки. Модель была построена для 16 каналов и в этом диапазоне показателя параллелизма коэффициент ускорения $K$ вышел на уровень 2,8 и далее не менялся. Полученные результаты моделирования означали приговор всему проекту. Поставленная цель – освоение массового динамического параллелизма средствами данного проекта не могла быть достигнута. Требовалось осмысление полученных результатов, анализ причин, определяющих эф-

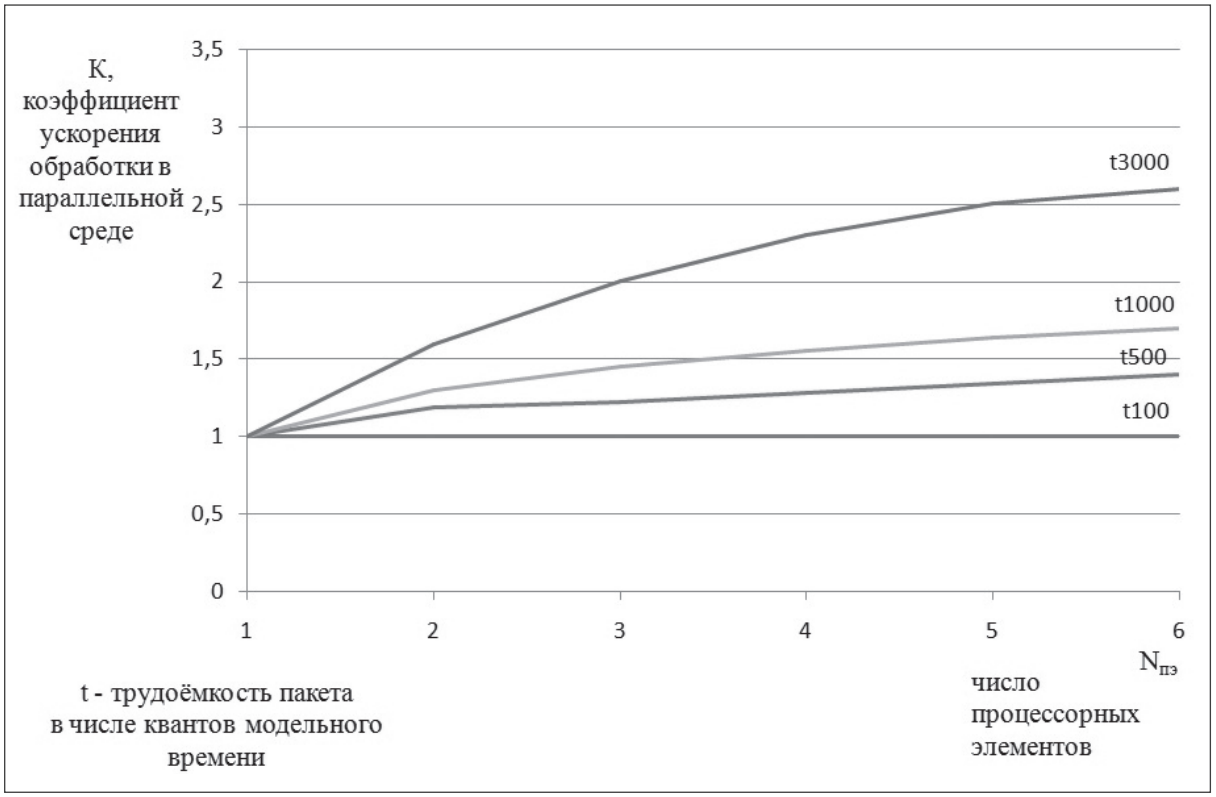


Рис. 4. Зависимость коэффициента ускорения K от числа каналов обработки N ко при разных значениях $t_n$ средней трудоёмкости пакетов

фект насыщения и глубокая ревизия подходов к организации вычислительных процессов. По данным личного общения авторы могут предполагать, что подобные результаты наблюдались и в других проектах с элементами Data Flow, однако широкого освещения в научной периодике эффект быстрого насыщения роста производительности в те годы не получил. Работа [18], на которую мы ссылаемся в данной статье была опубликована в виде препринта тиражом 200 экземпляров, а сам эффект насыщения в тексте завуалирован и понятен только участникам эксперимента. Мотивы такого поведения понятны и объясняются высокими рисками потерять финансирование проектов высокопараллельных систем. В тоже время конъюнктура предлагала другой путь – имелся определённый резерв роста ресурсных показателей вычислительных средств в рамках чисто технологических решений. Переход на новые значения технологических норм микроэлектронного производства обеспечивал устойчивый рост тактовой частоты, а, следовательно, и производительности вычислений.

## 4. Результаты замеров производительности при выполнении алгоритмов 3D БПФ

Проблема быстрой остановки роста производительности при наращивании степени параллелизма никуда не делась и со временем проявилась на макроуровне при эксплуатации кластеров и суперкомпьютеров. В этой связи целесообразно рассмотреть данные, полученные пользователями многопроцессорных систем в ходе исследования эффективности параллельного выполнения алгоритмов трёхмерного быстрого преобразования Фурье (3D БПФ), изложенные в [19]. Напомним, что существуют многочисленные версии специальных многопроцессорных систем, ориентированных на параллельное выполнение алгоритмов БПФ. Эти специализированные системы эффективно выпол-

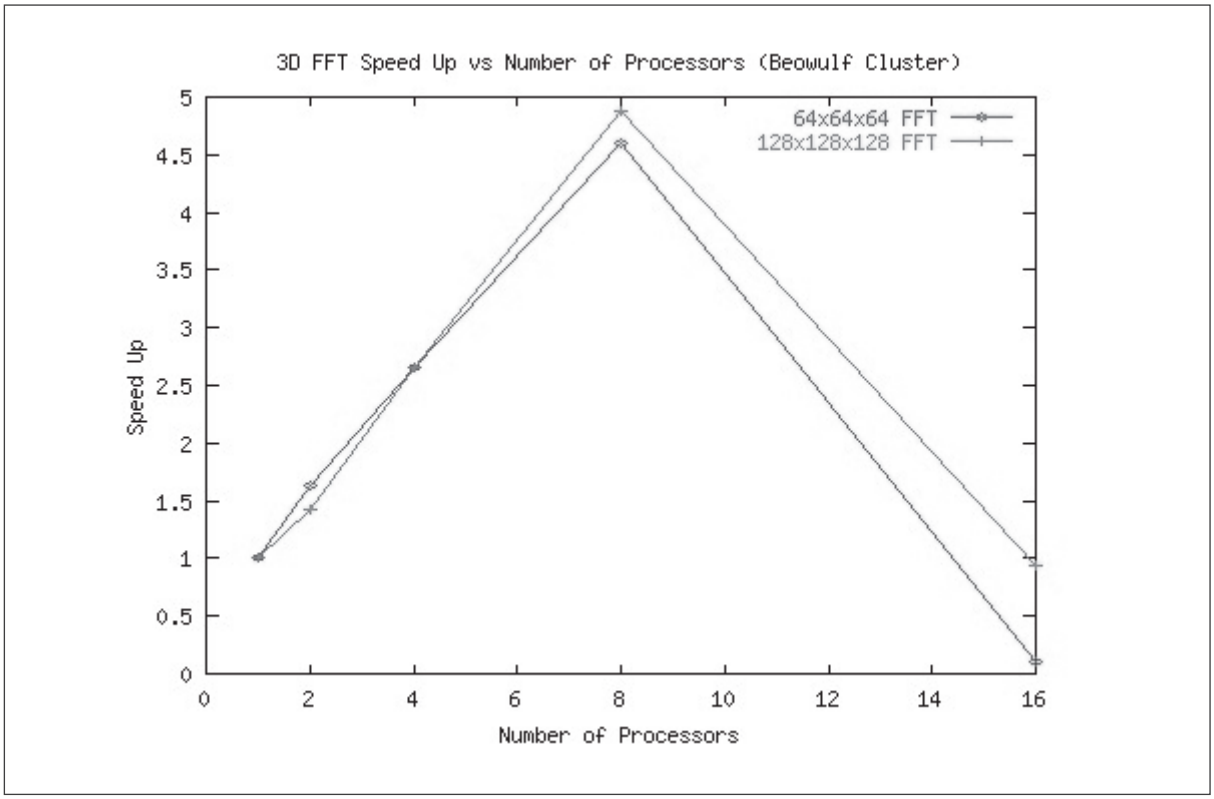


Рис. 5. Ускорения, полученные при прогоне алгоритмов 3D БПФ на установке Beowulf Cluster

няют только один алгоритм – БПФ. Другими алгоритмами их, как правило, не загружают. В тоже время алгоритмы БПФ являются наиболее проблемными при загрузке в универсальные параллельные системы. Именно алгоритмы БПФ характеризуются большими объёмами перекрёстных пересылок данных, в переходах, связывающих параллельные слои в общем графе алгоритма.

Мы приведём данные измерений временных затрат и вычисленные показатели ускорения в виде графиков из [19] и прокомментируем их.

Временные затраты получены в результате измерений, ускорения вычислены. В прогоне участвовали два варианта решения задачи - с размерностью 64x64x64 и с размерностью 128x128x128. Установка Beowulf Cluster это примитивный кластер, созданный на базе набора персональных компьютеров, объединённых средствами локальной офисной сети.

Вторая серия экспериментов выполнялась на установке Beowulf Cluster с более мощным коммутатором и более скоростной шиной. Результаты проведены на рис. 7 и рис. 8.

И наконец, авторы [19] получили доступ к более совершенной установке IBM SP2, построенной на базе скоростного коммутатора с развитой системой шин. Данные экспериментов приведены на рис. 9 и рис. 10.

Приведенная последовательность графиков показывает, что при прогоне алгоритмов с высоким потенциалом параллелизма коэффициенты ускорения довольно низкие и после прохождения пика резко падают. Ситуация заметно улучшается по мере улучшения обменной среды, поддерживающей межпроцессорные передачи.

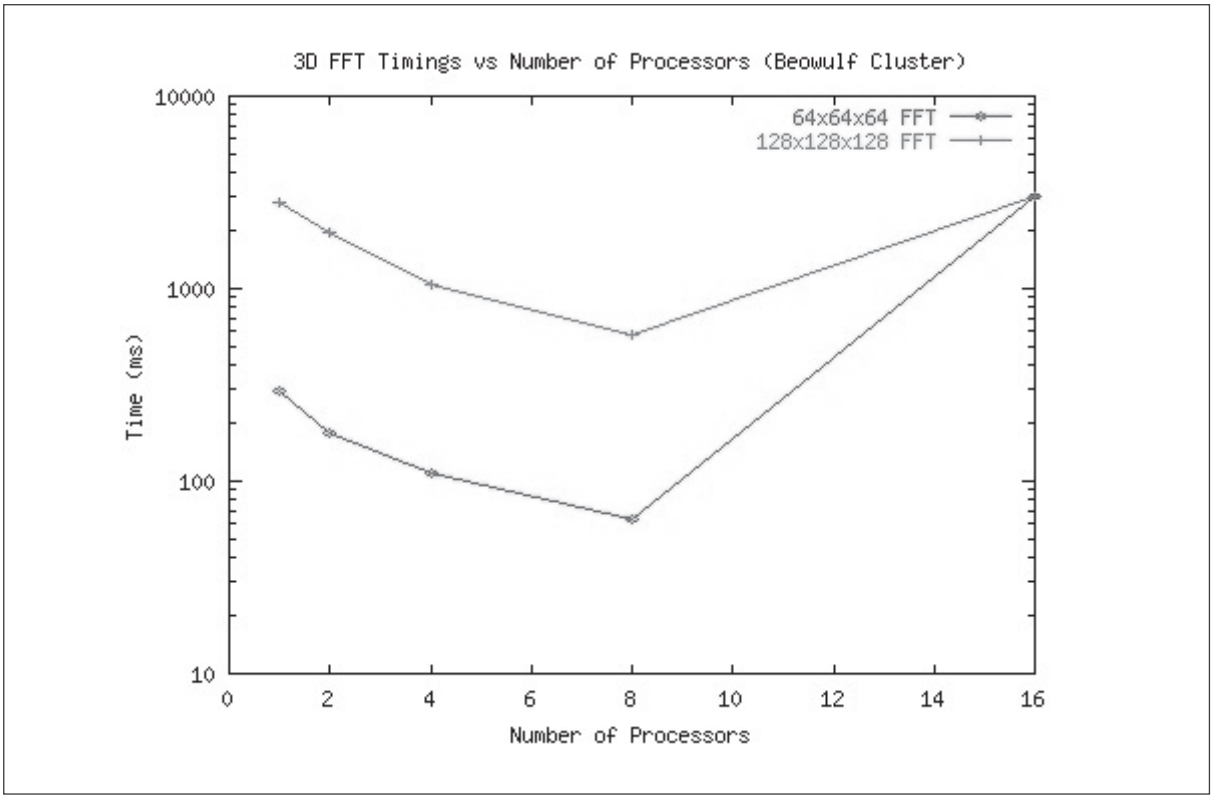


Рис. 6. Временные затраты при прогоне алгоритмов 3D БПФ на установке Beowulf Cluster

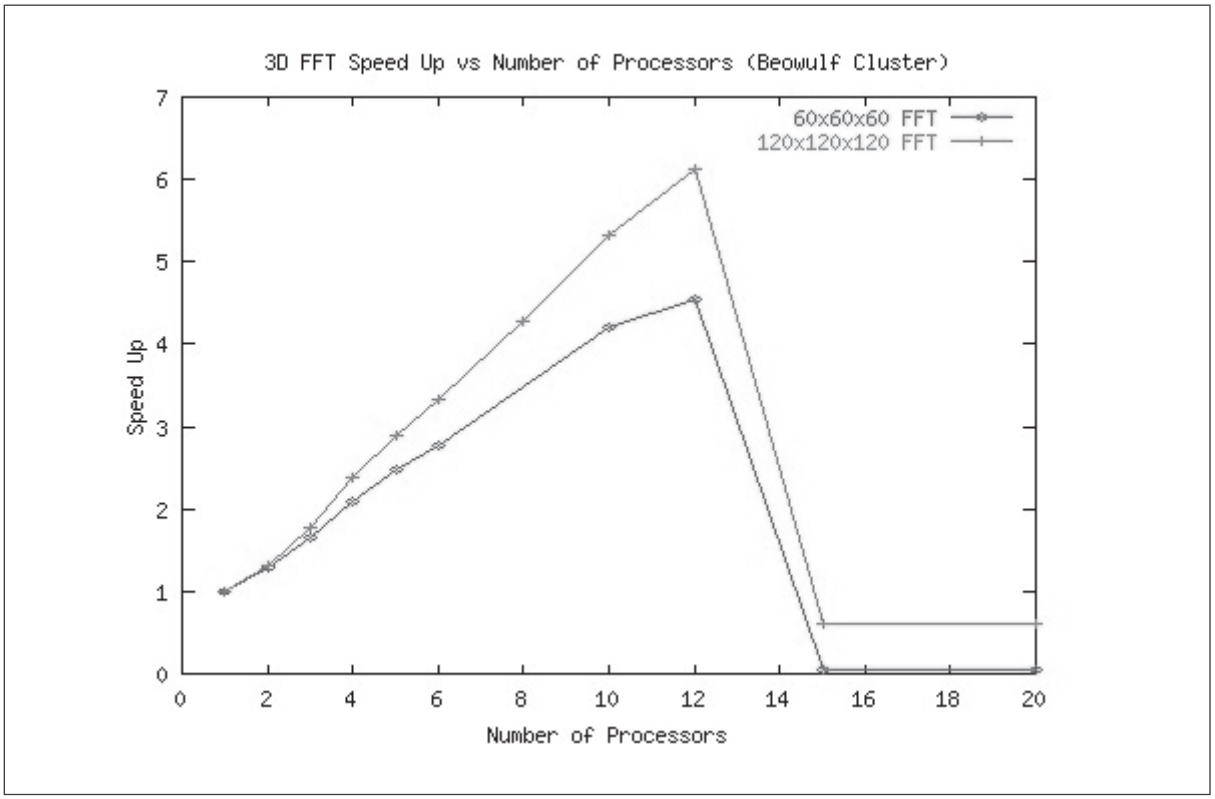


Рис. 7. Ускорения, полученные при прогоне алгоритмов 3D БПФ на улучшенной установке Beowulf Cluster

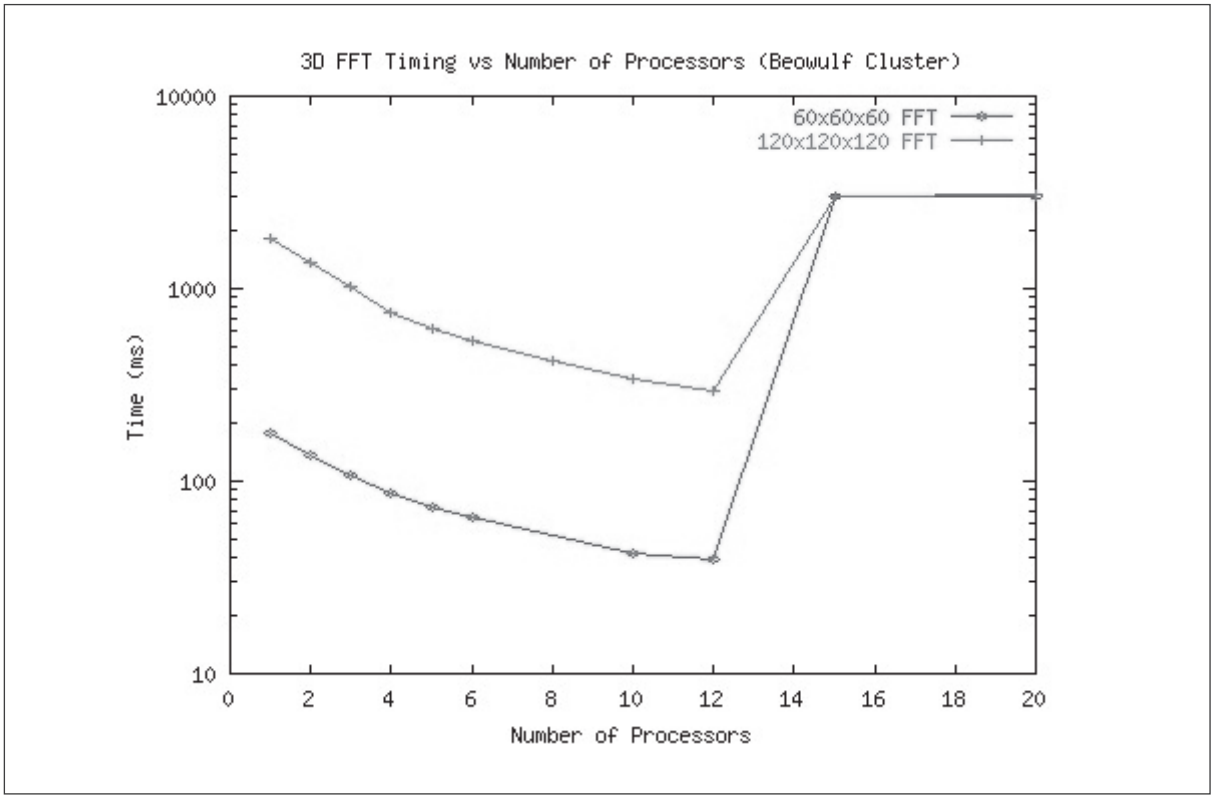


Рис. 8. Временные затраты при прогоне алгоритмов 3D БПФ на улучшенной установке Beowulf Cluster

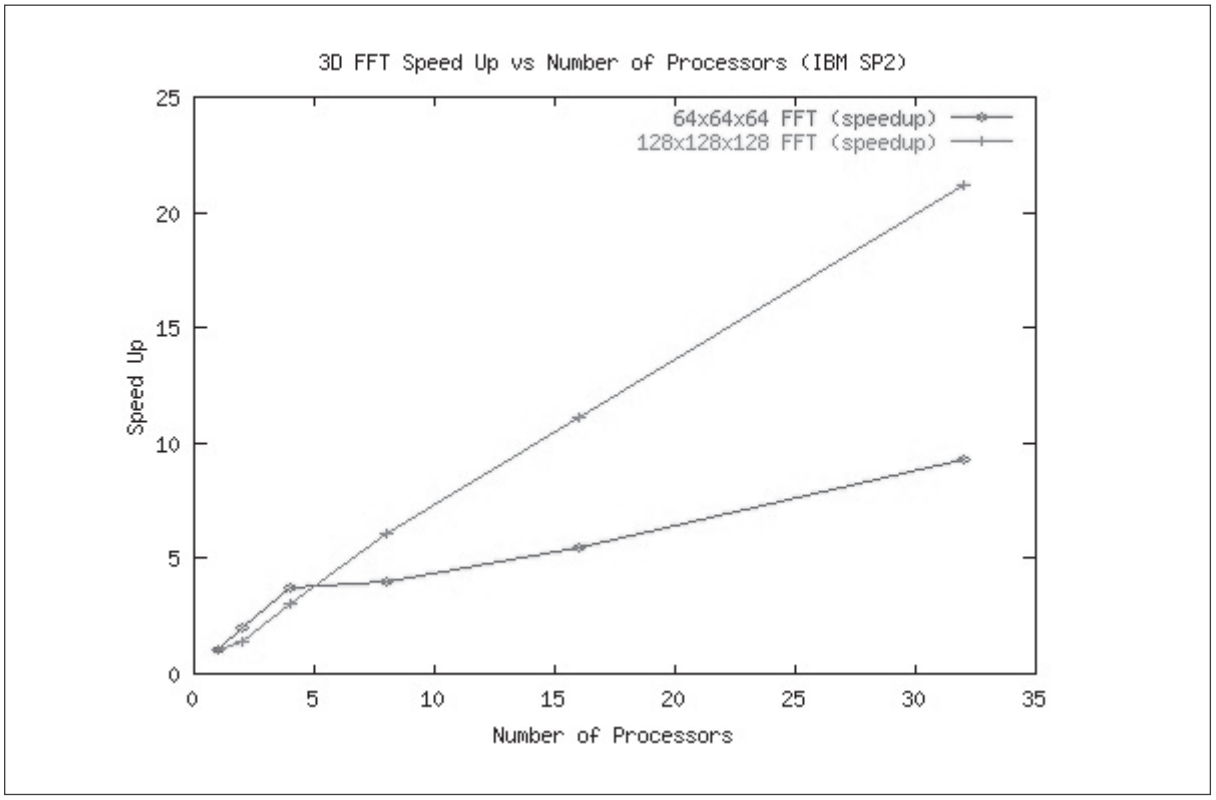


Рис. 9. Ускорения, полученные при прогоне алгоритмов 3D БПФ на установке IBM SP2

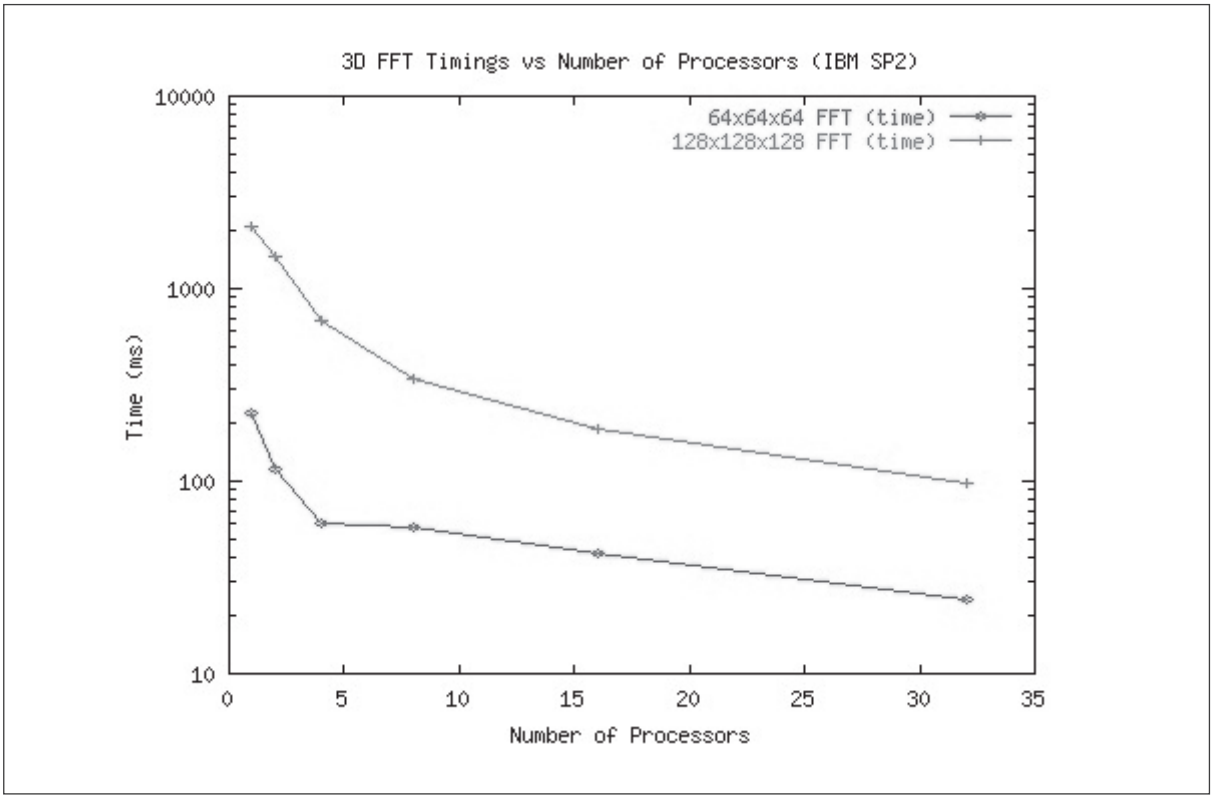


Рис. 10. Временные затраты при прогоне алгоритмов 3D БПФ на установке IBM SP2

## 5. Анализ факторов, ограничивающих рост производительности параллельных систем

Подведём итог. Рассмотрены ситуации роста производительности в двух достаточно разных средах – в классической многопроцессорной системе с общей шиной и в перспективном проекте неклассической архитектуры с элементами Data Flow. При этом результаты экспериментов похожие, если не сказать подобные. В случае с редукционно-потоковой машиной рассматривался результат модельного эксперимента, в случае с классической структурой получены данные измерений натурного эксперимента. В обоих случаях ход графиков роста коэффициента ускорения очень похож на графики, иллюстрирующие проявления закона Амдала. Однако надо сразу отметить, что наблюдаемый нами эффект быстрого прекращения роста производительности к закону Амдала не имеет никакого отношения, поскольку в обоих случаях насыщение наступает задолго до исчерпания потенциала параллелизма задачи. Мы имеем дело с другим явлением, природу и физический смысл которого нам предстоит выяснить.

Построим модель распределения временных затрат при параллельных вычислениях на конкретном примере реализации алгоритма быстрого преобразования Фурье (БПФ). На рис. 11 приведен граф процесса вычисления алгоритма 16-точечного БПФ. В данном случае размерность БПФ выбрана по соображениям наглядности и удобства визуального восприятия рисунка. Нам необходимо проиллюстрировать топологическую природу графа, представляющего регулярную сетку с определённым принципом построения. Далее известно, что граф процесса вычисления алгоритма БПФ легко масштабируется на любую размерность по степеням двойки.

Сплошными кружочками на рисунке обозначены комплексные операнды, записанные в алгебраической форме, каждый из которых представляются парой действительных коэффициентов. Нумерация операндов в левой части рисунка отображает порядок следования отсчётов во входной сигнальной последовательности. На выходе происходит перестановка порядка следования отсчётов, это побочный эффект данного алгоритма. Контурные кружочки обозначают базовую процедуру с двумя входами и двумя выходами, называемую бабочкой БПФ. Базовая процедура представляет собой алгоритм решения системы четырёх алгебраических уравнений. На выходе базовой процедуры порождается пара комплексных чисел, каждое из которых представлено парой действительных коэффициентов. Мы намеренно опускаем целый ряд деталей с целью установления основных свойств графа вычисления БПФ. Для нас важно, что граф носит регулярный характер. На рис. 9 видно, что структура графа разбивается на ряд вертикальных слоёв, представленных одним числом вычислительных процедур. Вся структура графа заполнена одной базовой вычислительной процедурой, которая принимает на входе и порождает на выходе одни и те же форматы данных. Параллелизм обработки данных осуществляется в пределах слоя, слои образуют последовательность, при переходах между слоями осуществляется обмен данными. Фазы параллельной обработки и фазы обменов данными идентичны по всему графу, что позволяет корректно провести подсчёт временных затрат на осуществление вычислительных и обменных операций.

Размерность алгоритма наращивается по степеням двойки и если число входных отсчётов равно $N$, то число слоёв в графе равно $\log_2 N$, а число вычислительных процедур в слое равно $\frac{N}{2}$. Соответственно общее число процедур в графе равно $\frac{N}{2}\log_2 N$. Основные показатели структуры графа, необходимые для вычисления производительности параллельного представления процесса масштабируются, что позволяет нам воспользоваться наглядностью графического представления 16-точечного БПФ и далее пересчитать требуемые характеристики для любой другой размерности. Для 16-точечного БПФ имеем следующие показатели число слоёв в графе равно 4, число базовых проце-

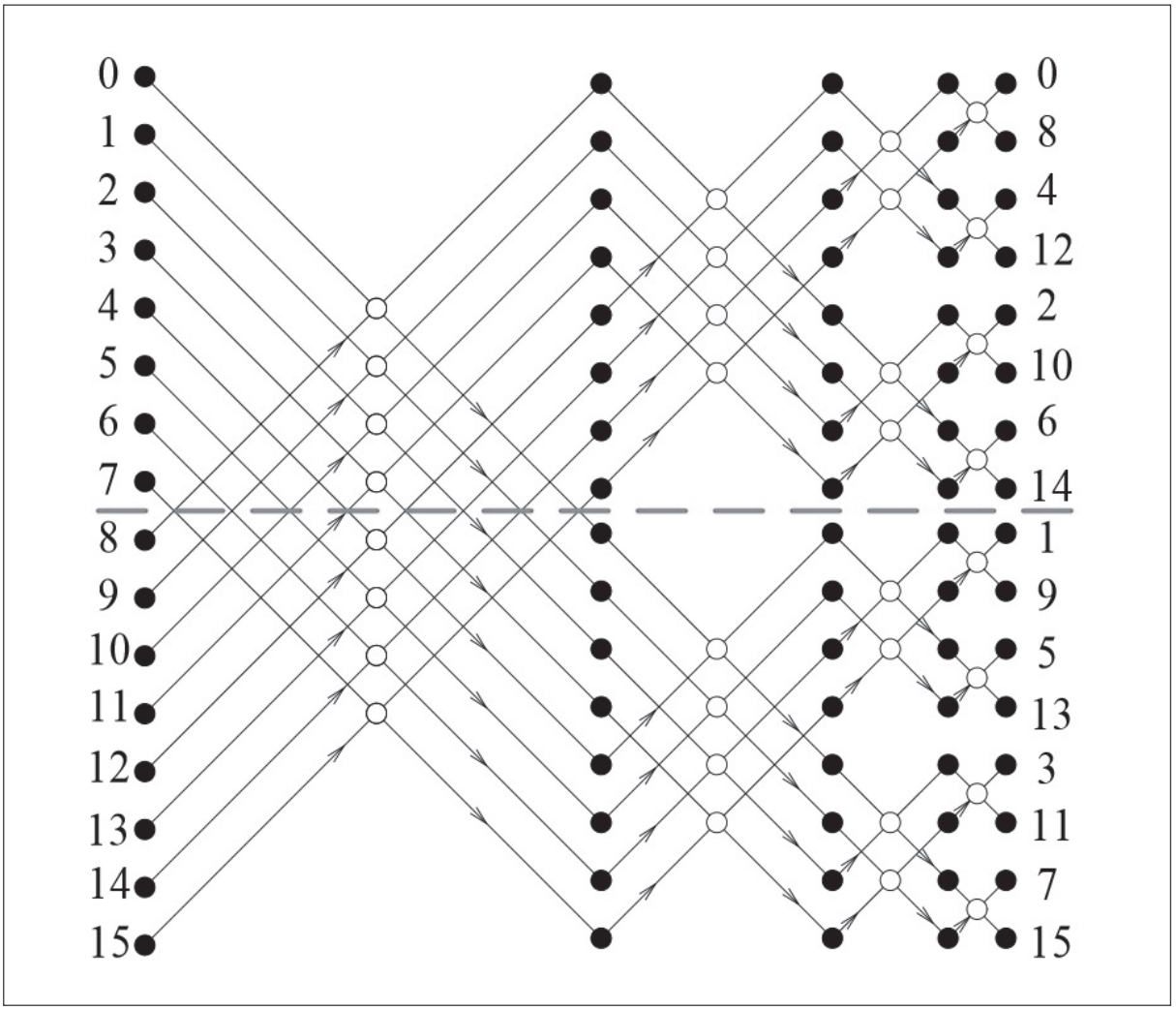


Рис. 11. Граф процесса вычисления 16-точечного БПФ .

дур в слое равно 8, всего процедур в графе 32. Наша задача состоит в том, что бы построить модель вычисления коэффициента ускорения, в которой в явном виде учитываются время вычисления базовых процедур и время передачи данных между процессорами при переходах с текущего слоя на следующий. При этом мы определяем время обмена данными как сумму времени передачи данных и времени выполнения протокольных событий, обеспечивающих доступ к обменной среде.

Определим ускорение $K$ как отношение времени выполнения алгоритма на одном процессоре $T_1(N)$ ко времени выполнения алгоритма на $n$ процессорах $T_n(N)$ , где $N$ размерность алгоритма, а $n$ - число процессоров. Поскольку искомое ускорение есть величина относительная для данной модели мы будем оценивать времена выполнения определённых действий в условных микротактах работы оборудования. Примем, что время выполнения базовой операции $b$ равно 10 микротактам. Тогда время выполнения 16-точечного БПФ на одном процессоре будет состоять только из времени выполнения базовых операций, поскольку обмен данными при этом отсутствует.

$$T_1(N) = b \cdot \frac{N}{2} \cdot \log_2 N = 10 \cdot 8 \cdot 4 = 320$$

Числитель будет состоять из трёх членов, зависящих от числа процессоров n . Время вычисления базовых процедур будет равно $\frac{320}{n}$. Или в общем виде

$$\frac{b \cdot \frac{N}{2} \cdot \log_2 N}{n}$$

Для определения времени пересылки данных необходимо рассмотреть структуру графа на рис. 9. При двух процессорах граф будет рассечен на две части по горизонтальному направлению как это отмечено пунктирной линией и две его половины будут загружены в разные процессоры. В этом случае обмен данными произойдёт только при переходе с первого на второй слой. При этом в каждом из двух процессоров будет вычисляться по 4 бабочки и каждая из них будет располагать только половиной входных данных, а вторую половину данных необходимо будет получить из другого процессора. Следовательно, обмен будет состоять из двух сеансов передач, содержащих по 4 отсчёта. Общий объём пересылаемых данных составит $\frac{N}{2}$ отсчётов.

При 4-х процессорах верхняя и нижняя половинки графа рассекаются аналогичным образом. В этом случае появляется необходимость обмена данными при переходе со второго на третий слой, объём пересылаемых данных по прежнему равен $\frac{N}{2}$ отсчётов, а суммарный объём составит $2 \cdot \frac{N}{2}$. Для 8 процессоров каждая выделенная по горизонтали часть графа вновь рассекается на две и мы получаем параллельное представление всего графа, состоящее из восьми последовательных нитей, каждая из которых содержит последовательность из трёх базовых операций. При этом появляется необходимость обменов при переходе со второго на третий слой с тем же объёмом передаваемых данных. Общий объём обменов будет равен $3 \cdot \frac{N}{2}$. отсчётов. Таким образом, в общем случае число передаваемых отсчётов будет равно $\frac{\mathrm{N}}{2} \cdot \log_2 N$ .

Протокольные события в обменной среде сопровождают сеансы связи и в упрощённой ситуации их подсчёт можно осуществить как некоторое фиксированное время, выраженное в условных микротактах и привязанное к сеансу связи. По мере наращивания числа процессоров происходит увеличение числа сеансов и уменьшение объёмов данных, передаваемых в каждом сеансе. При двух процессорах происходит один обмен па переходе от первого слоя ко второму, который состоит из 2 сеансов – от первого процессора ко второму и от второго к первому. При четырёх процессорах имеет место 2 обмена, в каждом из которых осуществляется по 4 сеанса и общее число сеансов равно 4 + 4. При восьми процессорах происходит три обмена и в каждом из них по 8 сеансов, всего 8 + 8 + 8. Таким образом, в общем случае число сеансов может быть определено как $n \cdot \log_2 n$.

Посчитаем общие затраты времени при 2 процессорах. Время вычисления сократится вдвое и составит $\frac{320}{2} = 160$ условных микротактов.

Время передачи данных учитывается как объём передаваемых данных приведенный к пропускной способности шины. При двух процессорах осуществляется передача восьми отсчётов по два слова в каждом, что равно 16 словам. По определению мы принимаем производительность шины равную передаче 1 слова за один микротакт, следовательно, искомое время обмена данными составит 16 условных микротактов.

Временные затраты на осуществление протокольных событий вычисляется как затраты на один протокол умноженные на число сеансов. Примем по определению, что на один протокол затрачивается 5 условных микротактов. Для данного случая, осуществления одного обмена, состоящего из двух сеансов, это составит $2 \cdot 5 = 10$.

Таким образом, суммарные временные затраты на вычисление 16-точечного БПФ В нашей модели при двух процессорных элементах составляют 186 условных микротактов, а коэффициент ускорения $K$ = 1,72. Аналогично проводится подсчёт для 4 и 8 процессоров. Данные сведены в Таблицу 1.

График роста коэффициента ускорения для 16-точечного БПФ представлен на рис. 12.

Приведенная модель в целом поддерживает динамику роста коэффициента ускорения, качественно совпадающую с результатами наблюдений и модельных экспериментов, описанных ранее. Модель может быть аппроксимирована на любую размерность алгоритма БПФ. Общая формула для подсчёта времени выполнения алгоритма $T_n(N)$ приводится ниже.

$$T_n(N) = \frac{b \cdot \frac{N}{2} \cdot \log_2 N}{n} + \frac{N \cdot \log_2 n}{\alpha} + \beta \cdot n \cdot \log_2 n$$

Таблица 1

| Число процессоров $n$ | **1** | **2** | **4** | **8** |
|---|---|---|---|---|
| Время вычисления (условные микротакты) | 320 | 160 | 80 | 40 |
| Время пересылки данных (условные микротакты) | | 16 | 32 | 48 |
| Время выполнения протокола (условные микротакты) | | 10 | 40 | 120 |
| Суммарное время (условные микротакты) | 320 | 186 | 152 | 208 |
| Коэффициент ускорения $K$ | **1** | **1,72** | **2,11** | **1,54** |

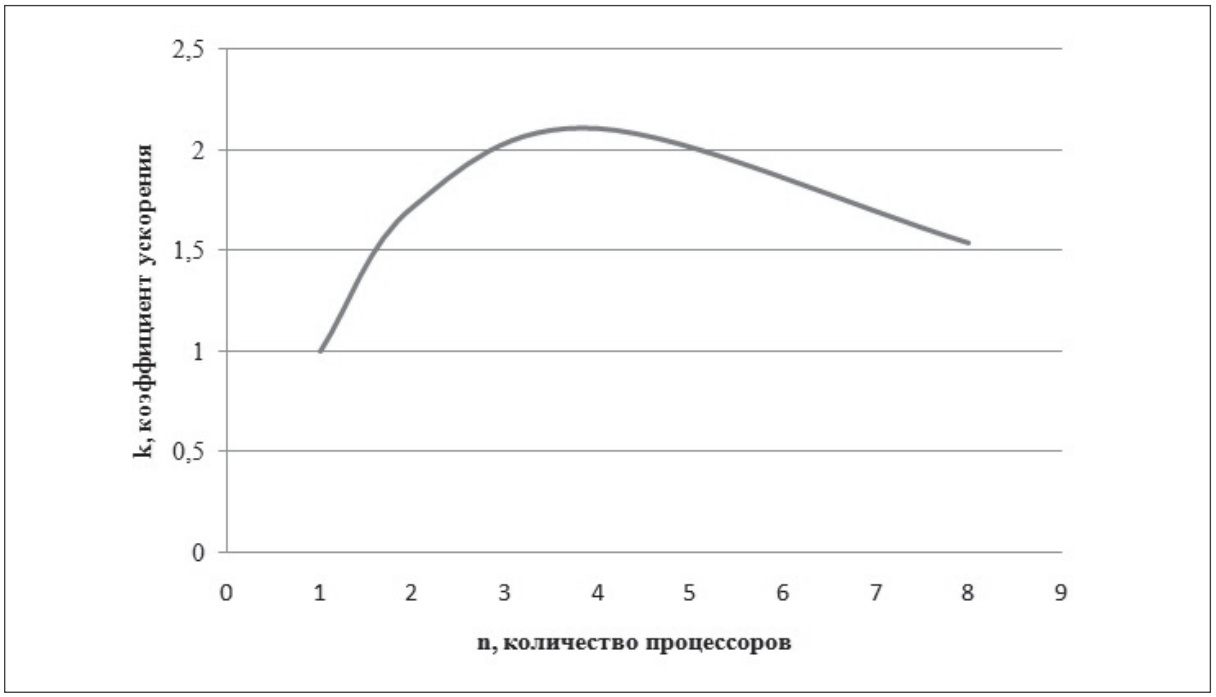


Рис. 12. График роста коэффициента ускорения для 16-точечного БПФ

Первое слагаемое – это соотношение для времени выполнения вычислений, которое учитывает время счёта базовой операции $b$ и число базовых операций в алгоритме равное $\frac{N}{2} \cdot \log_2 N$.

Второе слагаемое это время, затраченное на передачи данных между процессорами. Здесь объём передаваемых данных, выраженный в машинных словах, делится на величину $\alpha$, пропускную способность шины, которая в данном случае принимается равной 1, что означает передачу одного слова за один микротакт.

Третье слагаемое это затраты на осуществление протокольных мероприятий, обеспечивающих доступ к шине для совершения сеанса передачи данных. Параметр $\beta$ задаёт время выполнения протокола для одного сеанса. В данном примере мы определяем его значение равным 5 условным микротактам. Общее время затрат на протоколы вычисляется умножением $\beta$ на число сеансов, равное $n \cdot \log_2 n$.

В качестве демонстрационного примера мы выбираем результаты вычислений коэффициента ускорения для алгоритма БПФ на 1024 отсчёта. Граф алгоритма при данной размерности состоит из 10 слоёв по 512 бабочек в каждом слое, всего в графе содержится 5120 базовых процедур. Максимальный потенциал распараллеливания данного алгоритма составляет 512 последовательных линий из 10 базовых процедур каждая. Результаты вычислений для $N = 1024$ при $\alpha = 1$ и $\beta = 5$ приведены в Таблице 2.

График роста коэффициента ускорения приводится на рис. 13.

Таблица 2

| число процессоров $n$ | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|---|---|---|
| время выполнения (условные микротакты) | 51200 | 26634 | 14888 | 9592 | 7616 | 7520 | 8864 | 12048 | 18632 | 32356 |
| коэффициент ускорения $K$ | 1 | 1,92 | 3,44 | 5,34 | 6,72 | 6,81 | 5,78 | 4,25 | 2,75 | 1,58 |

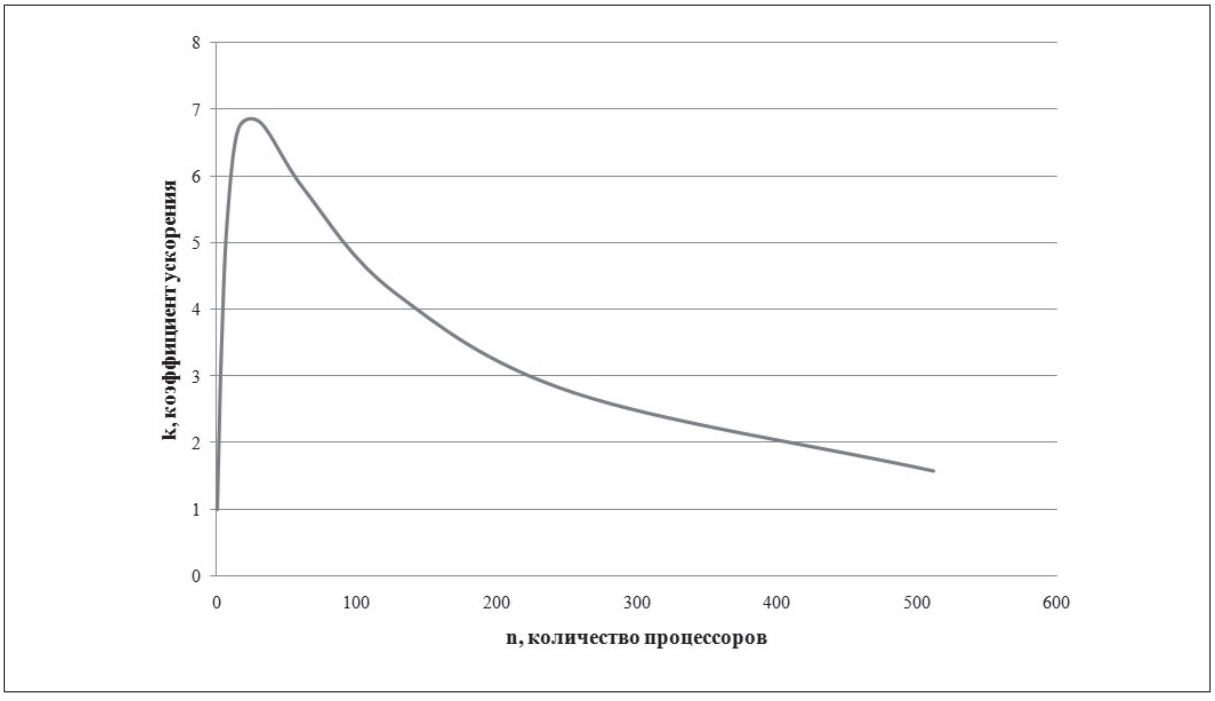


Рис. 13. График роста коэффициента ускорения для БПФ на 1024 точки

Наша модель позволяет исследовать поведение всех обозначенных компонентов, формирующих баланс временных затрат и их зависимость от числа процессоров. На рис.14 приводится динамика времени выполнения алгоритма по всем слагаемым: времени счёта, времени пересылки данных и времени протоколов доступа к обменной среде.

Приведенный график показывает полную картину во всём диапазоне значений числа процессорных элементов и временных затрат. В дополнение к общей картине на рис. 15 в увеличенном масштабе изображён начальный фрагмент динамики роста временных затрат.

Подведём итог - алгоритм имеет потенциал параллелизма, позволяющий распределить на 512 процессорах 512 последовательных линий, каждая из которых представляет собой последовательность из 10 базовых процедур. Следовательно, в идеальном случае можно получить коэффициент ускорения близкий к значению 512. Полученное на модели значение ускорения 6,81 , заметно отличается от ожидаемых 512. Модель позволяет обнаружить в явном виде причины такого расхождения. Время счёта устойчиво падает с ростом числа процессоров $n$, но его вклад в рост ускорения подавляется двумя слагаемыми, которые растут с ростом $n$. Это временные затраты на обмен данными и на выполнение протокольных событий. Обе растущие компоненты порождаются работой обменной среды и событиями, обслуживающими параллелизм.

Обменная среда является самой консервативной составляющей аппаратного обеспечения. Идея общей шины значительно упрощает структуру аппаратных средств, но в целом является инерционным звеном, снижающим показатели производительности вычислительных средств. Сделанные нами вычисления показывают неприемлемость применения общей шины при построении многопроцессорных систем с массовым параллелизмом.

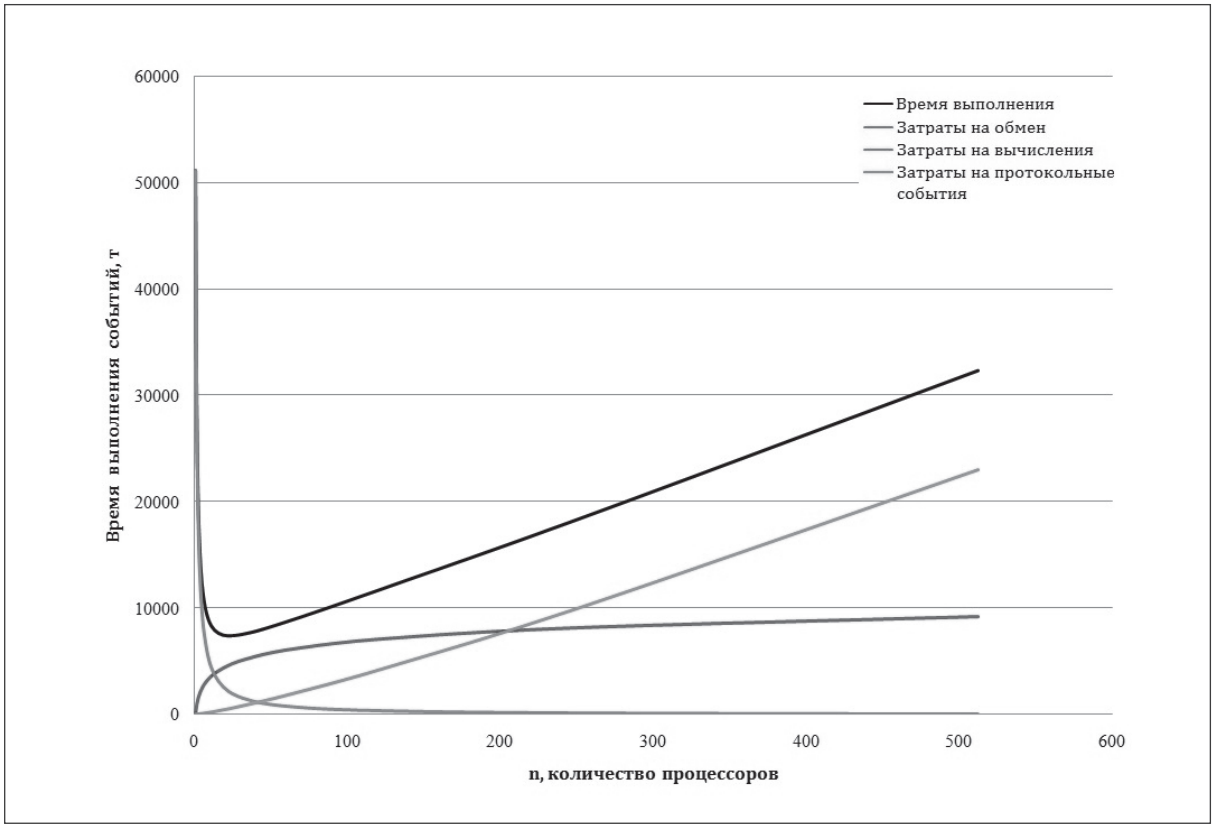


Рис. 14. Динамика времени выполнения алгоритма по основным слагаемым

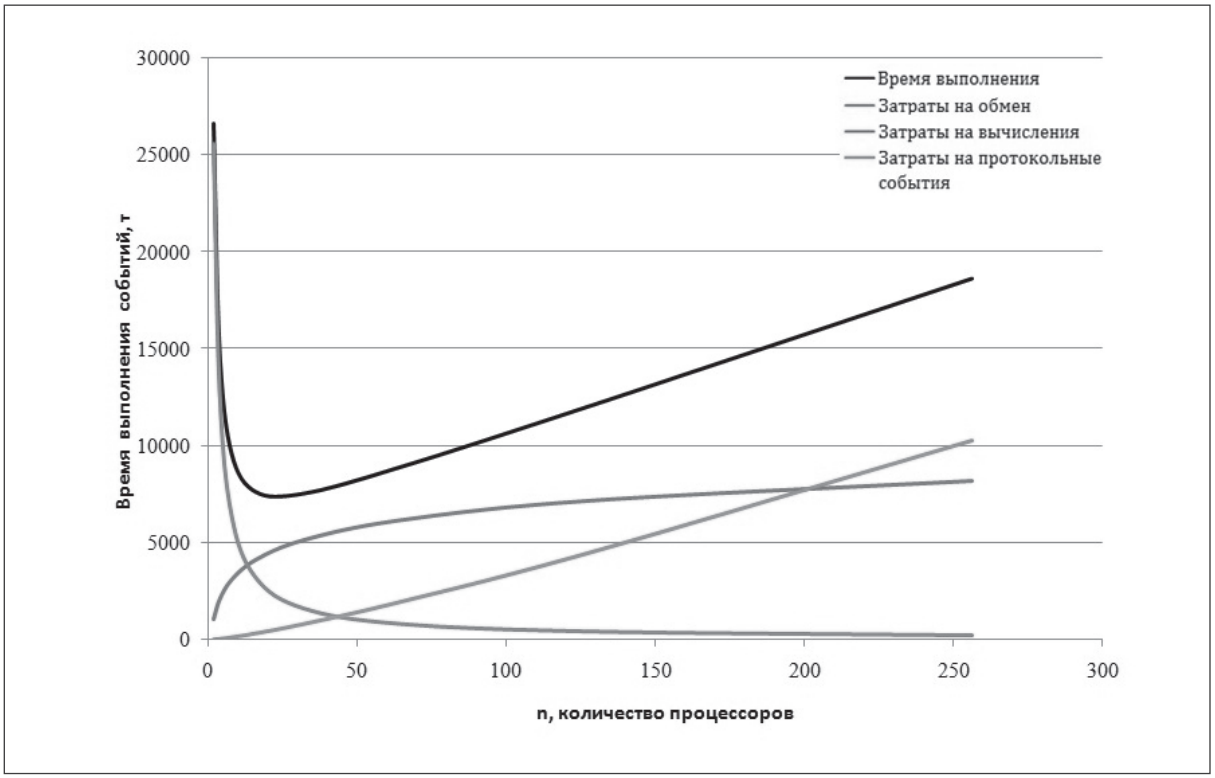


Рис. 15. Начальный фрагмент динамики времени выполнения алгоритма по основным составляющим

Далее мы рассмотрим улучшенный вариант применения общей шины. Допустим, что многоядерная структура формируется путём наращивания кластеров, состоящих из 16 процессоров, объединённых общей шиной. Это позволяет распараллелить обменные операции, локализованные в пределах кластера. При этом обмены между кластерами осуществляются по общей шине верхнего уровня, которая объединяет кластеры. Для модельных подсчётов сохраним ранее принятые параметры. Рассматривается граф алгоритма БПФ, $N = 1024$ при параметрах шины $\alpha = 1$ и $\beta = 5$, что означает передачу одного машинного слова за один такт и осуществление протокола доступа к шине за 5 тактов. Обмены между процессорами внутри кластера осуществляются одним проходом по шине кластера. Доступ процессора к обмену между кластерами требуют двух проходов по шинам. Сначала проход по кластерной шине для выхода на межкластерную шину и далее проход по этой шине в смежный кластер.

Время вычислений будет устойчиво снижаться с ростом $n$, а время выполнения обменов и протоколов будет рассчитываться по двум разным принципам. Общее время обменов в распараллеленной ветви будет делиться на число кластеров. В ветвях осуществления межкластерных обменов при суммировании будет учитываться факт двойного прохода по разным шинам. Результаты вычислений сведены в Таблицу 3.

На рис. 16 изображены кривые роста ускорения. Красным цветом отмечено ускорение в структуре с одной шиной, синим в системе с множеством шин.

На графике видно, что улучшение более чем скромное, переломить ситуацию быстрого насыщения роста производительности не удаётся. На рис. 17 приводится семейство кривых изображающих поведение отдельных компонентов процесса.

Красная линия, отражающая рост затрат на осуществление обменов данными в начальной фазе демонстрирует попытку сформировать тенденцию к снижению затрат, но далее растущий параллелизм подавляет эту тенденцию.

Таблица 3

| процессоры | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|---|---|---|
| Вычисление | 51200 | 25600 | 12800 | 6400 | 3200 | 1600 | 800 | 400 | 200 | 100 |
| Передача | 0 | 1024 | 2048 | 3072 | 4096 | 4096 | 5120 | 6656 | 8346 | 10368 |
| Протокол | 0 | 10 | 40 | 120 | 320 | 640 | 1600 | 4160 | 10560 | 25820 |
| Сумма | 51200 | 26634 | 14888 | 9592 | 7616 | 6336 | 7521 | 11216 | 19106 | 36288 |
| Ускорение на 1 шине | 1 | 1,92 | 3,44 | 5,34 | 6,72 | 6,81 | 5,78 | 4,25 | 2,75 | 1,58 |
| Ускорение на многошинной структуре | 1 | 1,92 | 3,44 | 5,34 | 6,72 | 8,08 | 6,8 | 4,56 | 2,68 | 1,4 |

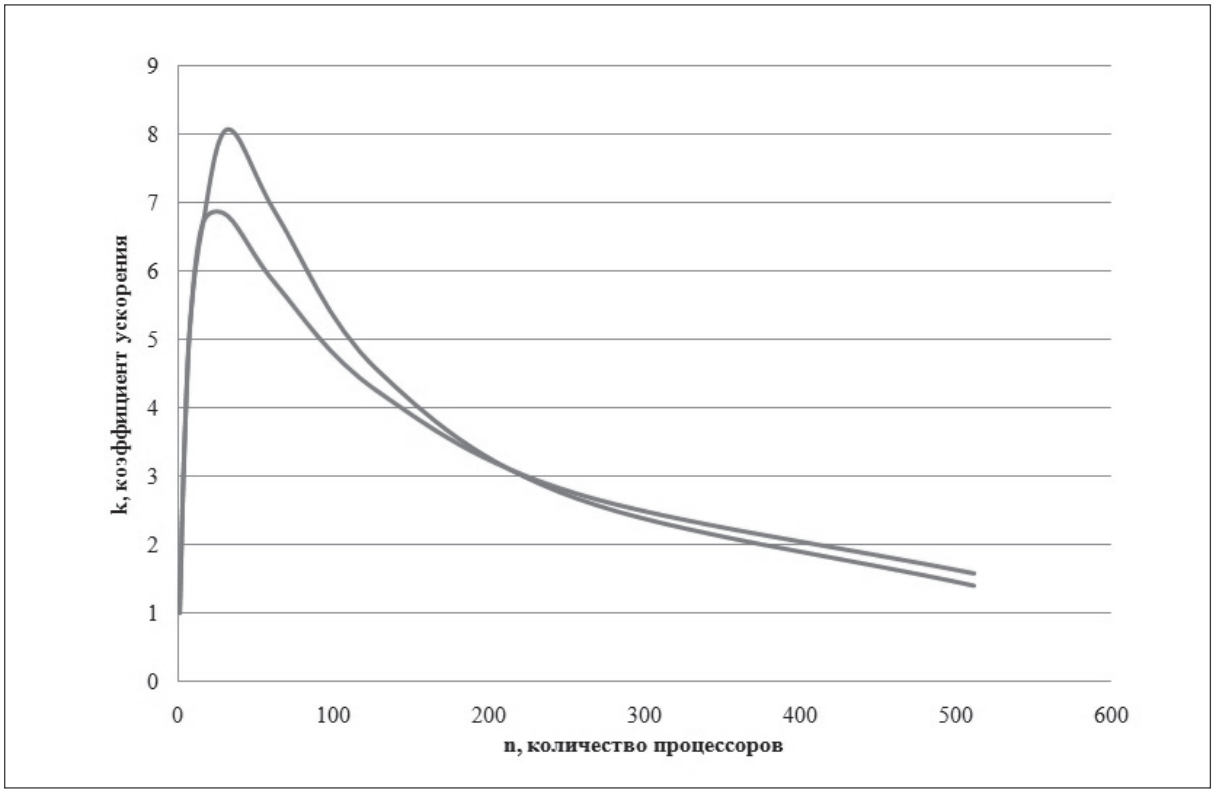


Рис. 16. Кривые роста ускорения. Красная линия для одной шины, синяя для множества шин

Данные модельного эксперимента показывают, что структура с множеством шин может дать определённое улучшение на относительно небольших значениях параллелизма, в пределах нескольких десятков. Далее по мере роста числа процессоров в область сотен экземпляров результаты многошинной структуры сравниваются с результатами одной шины. Таким образом, мы можем объяснить заметное улучшение результатов, полученных в [19] . В последнем эксперименте вычисления проводились на более совершенной установке IBM SP2 с более скоростной многошинной обменной средой.

На рис. 17 видно, что время, затраченное на пересылки данных растёт умеренно и при больших значениях параллелизма роль этой компоненты стабилизируется. Наиболее агрессивной компонентой, определяющей быстрый рост временных затрат при больших значениях параллелизма является время выполнения протокольных событий. Дело в том, что с ростом числа процессоров обмен данными дробится на множество мелких порций, каждая из которых оформляется как самостоятельный сеанс передачи, требующий доступа к шине. Таким образом число протоколов разрастается и их удельный вес в обменных операциях становится подавляющим.

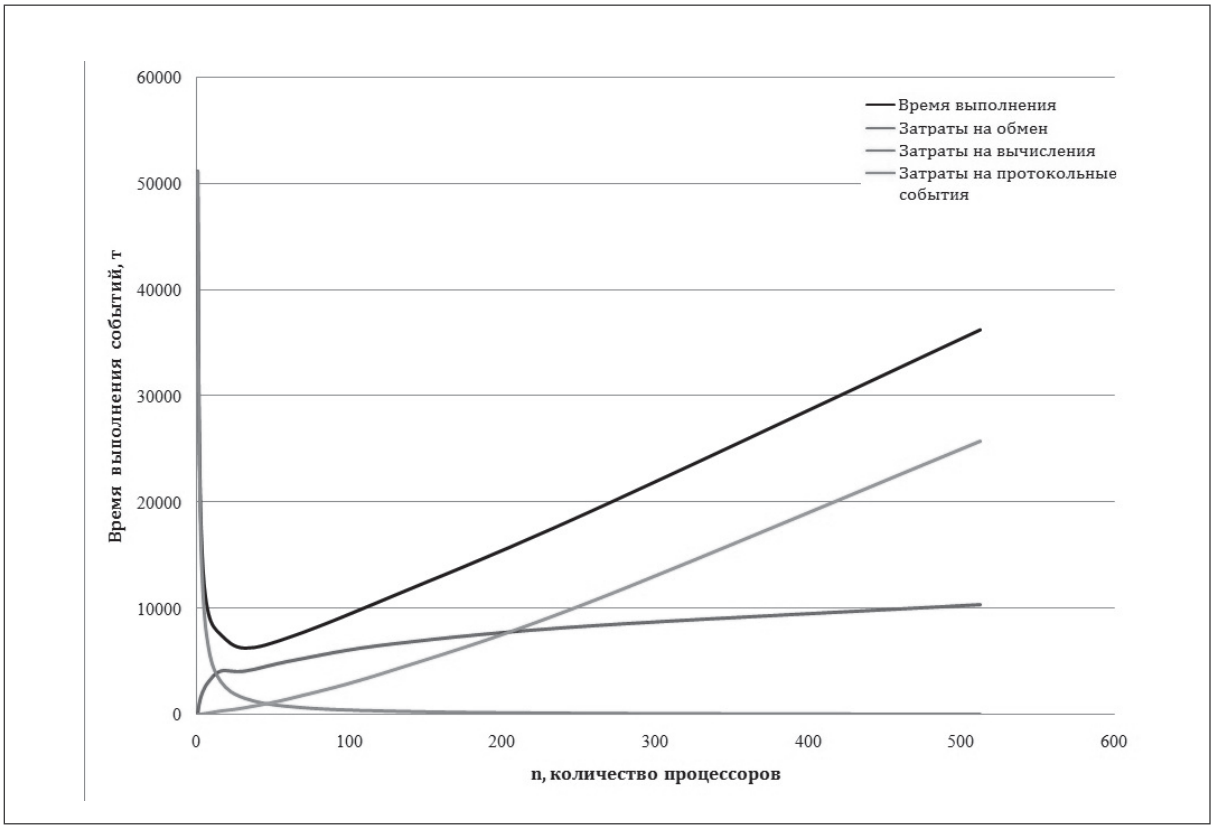


Рис. 17. Динамика роста временных затрат по основным компонентам процесса

Следует учесть, что в нашей модели приняты допущения о некой идеальной ситуации абсолютной доступности шины. В реальности при коллективном использовании шины происходят конфликты доступа, при которых запускаются процедуры шинного арбитража и ожидания в очереди. Фактическая ситуация значительно хуже. Ценность данной модели в том, что она позволяет выявить основные причины подавления роста производительности на качественном уровне.

## Заключение

Тема настоящей статьи – проблемы освоения массового динамического параллелизма вычислений и построение высокопараллельных компьютерных систем. Во введении приведены данные о современном состоянии технологии микроэлектронного производства и ближайших перспективах его развития. Основная проблема заключается в том, что современная технология позволяет строить многопроцессорные системы, в которых степень параллелизма достигает десятков тысяч параллельных каналов обработки и подтверждена тенденция к её дальнейшему росту. В то же время практика эксплуатации многопроцессорных систем при решении ряда актуальных и широко распространённых задач обнаруживает эффект быстрого насыщения и остановки роста производительности при очень малых показателях коэффициента ускорения. Задачи, имеющие потенциал параллелизма порядка сотен и тысяч независимых событий в реальности достигают ускорения не боле чем в 7 – 8 раз при 10 – 15 процессорах. В наиболее удачных реализациях достигается ускорение в несколько десятков раз, что соответствует активному использованию не более чем 10% возможностей аппаратуры.

Рассмотрены результаты натурных и модельных экспериментов, которые описывают данный эффект но не позволяют достаточно полно исследовать причины его воз-

никновения. Построена модель, описывающая динамику роста временных затрат по основным компонентам процесса Модель построена для конкретного алгоритма быстрого преобразования Фурье. Граф алгоритма носит регулярный характер и легко масштабируется по степеням двойки, что значительно облегчает счёт и вывод соотношений. Результаты расчётов по данной модели позволяют определить критические компоненты процесса, подавляющие рост производительности при росте параллелизма. Модель носит частный характер и построена для одного конкретного алгоритма, однако, выявленные на ней тенденции носят общий характер и позволяют анализировать ситуацию в целом. Проблема заключается в том, что при построении параллельных систем проявляются специфические затраты на обслуживание параллелизма, которые растут опережающим темпом и быстро подавляют ожидаемый рост производительности. В условиях, когда технология предоставляет возможности реализации массового параллелизма решение данной проблемы более чем актуально.

Детальный анализ явлений подавляющих рост производительности в параллельных системах и обсуждение путей их устранения излагаются во второй части данной статьи.

## Литература


1. *Шайтан К.В., Антонов М.Ю., Шайтан А.К., Новоселецкий В.Н., Боздагонян М.Е., Касимова М.А.* Метод молекулярной динамики в исследованиях свойств биологических объектов // Наноструктуры. Математическая физика и моделирование, 2012, 6 (1,2), 63-79.
2. *Блинов В.Н., Совенюк А.А.* Программирование задач физики конденсированного состояния с использованием MPI // Наноструктуры. Математическая физика и моделирование, 2012, 7 (1), 5-107.
3. *Викул Е.А., Тужилин А.А.* Геометрия аминокислот и полипептидов: случай рентгеноструктурного анализа // Наноструктуры. Математическая физика и моделирование, 2014, 11 (2), 5-29.
4. *Григоренко Б.Л., Князева М.А., Исаев Д.А., Новичкова А.В., Немухин А.В.* Компьютерное моделирование химических реакций в сложных биологических системах // Наноструктуры. Математическая физика и моделирование, 2014, 10 (2), 95-107.
5. *Блинов В.Н.* Дальнодействующие взаимодействия в компьютерном моделировании систем в конденсированном состоянии // Наноструктуры. Математическая физика и моделирование, 2014, 10 (1) 5-29.
6. *Ожигов Ю.И.* Представление декогерентности при компьютерном моделировании квантовых состояний наносистем // Наноструктуры. Математическая физика и моделирование, 2013 8 (1) 47-65.
7. Обзор продуктов семейства Tesla Kepler // NVIDIA.RU: NVIDIA Tesla GPU Accelerators, сайт производителя. URL: http://www.nvidia.ru/content/tesla/pdf/NVIDIA-Tesla-Kepler-
8. *Черняк Л.* Многоядерные процессоры и грядущая параллельная революция.// Открытые системы 2007 4, 33-42.
9. *Воеводин В.В*. Математические основы параллельных вычислений // ННГУ 2009. 583 стр.
10. *Sidlausakas D.* Lecture on multicores // Danish National Research Foundation, Madalgo, 12.03.2013, p. 8. URL: http://www.cs.au.dk/~gerth/ae13/slides/multicores.pdf
11. *Gene M. Amdahl*. Validity of the single processor approach to achieving large scale computing capabilities // IBM Sunny vale, California. AFIPS spring joint computer conference. 1967. URL: http://www-inst.eecs.berkeley.edu/~n252/paper/Amdahl.pdf
12. *Dennis J. B.* First version of Data Flow procedure language // Lecture Notes in Computer Science 1974, 363 – 376.
13. *Городня Л.Г.* Основы функционального программирования // М., 2004 280 ст.
14. *Амамия М., Танака Ю.* Архитектура ЭВМ и искусственный интеллект // Мир, Москва, 1993 397 стр.
15. *Галушкин А.И., Точенов В.А.* Транспьютерные системы – начало становления в России ЭВМ с массовым параллелизмом // Нейрокомпьютер, , 2005 3, 22-38
16. *Backus J.* Can Programming Be Liberated from von Neumann Style? A Functional Style and Its Algebra of programs // 1977 Turing Award Lecture, P. 63-130 http://rkka21.ru/docs/turing-award/jb1977r.pdf http://rkka21.ru/docs/turing-award/jb1977e.pdf
17. *Махиборода А.В. и другие,* Система потоковой обработки информации с интерпретацией функциональных языков // Авторское свидетельство № 1697084 опубликовано 29. 06. 1989 г. Положительное решение 25.05 1990 г.

18. *Соколовский Ю.Л.* Предпроектное исследование персональной ЭВМ редукционно-потокового типа средствами имитационного моделирования // Препринт 92-17, Институт кибернетики АН УССР, Киев 1992 30 стр.
19. *Cramer C. Board J.* The Development and Integration of Distributer 3D FFT for a Cluster of Workstations // «4th Annual Linux Showcase & Conference, Atlanta October 10-14 2000», Pp. 121–128 of the *Proceeding*;

# PROBLEMS OF IMPLEMENTATION OF MASSIVE DYNAMIC PARALLELISM. I

A.V. Makhiboroda, A.V. Ilichev, A.A. Podobin

*Department of Applied Mathematics MIEM,*

*National Research University "Higher School of Economics"*

makhiboroda@yandex.ru

Received 01.11.2015

Modern state of microelectronic production technology allows to place thousands of processing elements on a single chip and the trend toward sustainable growth of this indicator over the next decade is declared. The increase in clock frequency is stopped and fixed at the optimal value of 2.2 GHz. The development of massive parallelism becomes the main instrument for ensuring the growth of computing resources. However, there are a number of fundamental issues to be resolved to the realization of massively parallel computing. The article describes a number of examples illustrating the effect of a rapid stop of performance growth while increasing values of parallelism. The reasons for suppressing the performance growth in the initial steps of parallelism growth are analyzed. Measures on the transfer of the saturation point of growth to higher values of the number of processor elements are proposed. The directions of architecture development and principles of algorithms with the prospect of effective implementation of the massive dynamic parallelism are discussed.