



Е. А. Еремин,

Пермский государственный гуманитарно-педагогический университет

ИЗУЧАЕМ КОМПЬЮТЕРНУЮ АРХИТЕКТУРУ: ОТ ОДНОПРОЦЕССОРНЫХ МАШИН К МНОГПРОЦЕССОРНЫМ СИСТЕМАМ*

Аннотация

Статья посвящена методике знакомства с многопроцессорной архитектурой. При изложении теории прослеживается логика развития от обычных компьютеров с одним процессором к многопроцессорным вычислительным системам. Для практических занятий разработаны и реализованы программы-имитаторы, демонстрирующие решение вычислительной задачи на системах с общей и распределенной памятью.

Ключевые слова: архитектура, процессор, память, многопроцессорная система, система с общей памятью, система с распределенной памятью, параллельные вычисления, учебные программы.

Контактная информация

Еремин Евгений Александрович, канд. физ.-мат. наук, доцент, доцент кафедры мультимедийной дидактики и ИТО Пермского государственного гуманитарно-педагогического университета; адрес: 614990, г. Пермь, ул. Сибирская, д. 24; телефон: (342) 238-63-31; e-mail: eremin@pspu.ru

Е. А. Eremin,
Perm State Humanitarian Pedagogical University

LEARNING COMPUTER ARCHITECTURE: FROM SINGLE-PROCESSOR MACHINES TO MULTIPROCESSOR SYSTEMS

Abstract

The article is devoted to the method of familiarization with a multiprocessor architecture. Reciting the theory, it traces the logic of evolution from an ordinary computer with single processor to a multiprocessor computing systems. The software imitators those demonstrate the solving of computational problem using common and shared memory systems are developed for practical lessons.

Keywords: architecture, processor, memory, multiprocessor system, system with common memory, system with shared memory, parallel computing, educational software.

Если отбросить детали и выделить общую идею построения большинства современных параллельных вычислительных систем, то останется лишь два класса [2].

При разработке программ для параллельных ЭВМ используются специализированные библиотеки, позволяющие организовать обмен данными между отдельными подзадачами и их синхронизацию. Для эффективной организации обмена данными необходимо знать, как происходит пересылка информации между процессорами, какова ее скорость и т. д. [5].

Введение

Практически во всех современных персональных компьютерах установлены многоядерные процессоры. **Многоядерный процессор** по сути представляет собой несколько процессоров, собранных внутри общего корпуса. Отсюда немедленно следует, что любой покупаемый сегодня в магазине компьютер содержит в себе несколько процессоров, и, следовательно, чтобы понять, как он работает, необходимо иметь хотя бы минимальные представления об особенностях многопроцессорных систем.

К сожалению, на указанное изменение устройства компьютеров наша учебная литература отреагировать еще не успела. В то же время отчетливо осознается необходимость в этом. «В школе уже формируется определенный интерес к тематике, связанной с параллельной обработкой данных. Разработка целостной методики пропедевтической подготовки учащихся основной школы в области параллельных вычислений — задача нескольких ближайших лет» [1].

В данной статье сделана попытка максимально просто описать основные идеи многопроцессорной архитектуры. При этом выбран понятный любому ученику способ перехода от одного процессора к нескольким путем ответа на естественный вопрос: «А что надо сделать, чтобы превратить обычный однопроцессорный ПК в многопроцессорный?»

В соответствии с выбранной стратегией, в статье сначала кратко напоминаются необходимые базовые принципы однопроцессорной архитектуры. После этого демонстрируется, как путем некоторых ее усовершенствований можно получить многопроцессорную систему. Причем, как оказывается, такие несложные общие рассуждения приводят к вполне реальным структурам, которые и реализованы на практике.

Когда новая архитектура придумана, возникает интерес к тому, насколько она эффективна. Для ответа на последний вопрос предлагается использовать специально разработанное учебное программное обеспечение, которое поможет оценить то ускорение вычислений, которое может быть получено при переходе от одного процессора к нескольким.

* Материалы к статье можно скачать на сайте ИНФО:
http://infojournal.ru/journals/school/school_02-2016/

1. Однопроцессорная архитектура

Прежде чем анализировать многопроцессорные системы, рассмотрим, как устроен обычный компьютер с одним процессором.

1.1. Что относится к архитектуре

Покажем сначала, что при подборе учебного материала важно четко понимать, что такое компьютерная архитектура и что к ней относится.

Понятие **архитектура ЭВМ** было введено в середине прошлого века известной фирмой IBM. В основе лежали коммерческие соображения. Предлагалось ввести понятие **семейства ЭВМ**, т. е. разных по сложности и стоимости моделей, одновременно полностью программно совместимых между собой. В результате крупные организации могли позволить себе купить дорогую модель семейства, а небольшие — приобрести самую дешевую. Важно, что программное обеспечение в обоих случаях было одинаковым, хотя время выполнения программы могло существенно отличаться.

Кроме термина «архитектура» существует еще один похожий термин — **структурная организация**. «Хотя точное определение этим понятиям дать довольно трудно, среди специалистов существует общепринятое мнение о смысле этих понятий и различии между ними» [8]. Термин **архитектура** относится только к тем характеристикам компьютерной системы, которые *доступны со стороны программы*. Термин же **структурная организация** подразумевает *конкретные устройства и их взаимосвязи*, реализующие заданные архитектурой принципы. Поясним разницу на примерах.

Пусть в систему команд компьютера входит операция умножения — это архитектура. А вот способ, каким арифметическое устройство у конкретной модели перемножает числа, — это структурная организация. Другими словами, программа, перемножающая два числа, на любой машине получит один и тот же ответ, но сделано это может быть совершенно по-разному и, соответственно, за разное время.

Еще один пример. Наличие оперативной памяти и способы обращения к ней из программы — архитектура. Конструкция микросхем памяти и их объем — структурная организация.

Наверно, у вас возник вопрос: а зачем нам все это надо знать? Оказывается, при отборе материала для изучения это необычайно полезно! Архитектура консервативна и содержит наиболее важные характеристики, значит, для понимания общей картины *изучать надо именно то, что относится к архитектуре*. Запомним это.

Сформулируем два правила, которые легко позволят на практике определять, что относится к архитектуре, а что нет. По своей сути это следствия из рассмотренного выше определения.

Правило 1 (П1). То, чем может управлять программа, относится к архитектуре, все остальное — нет.

Правило 2 (П2). Если при отсутствии некоторого устройства любая программа продолжает работать (пусть и медленно), данное устройство к архитектуре не относится.

Правила П1 и П2 пригодятся нам в дальнейшем при построении максимально простой картины устройства компьютера.

1.2. Однопроцессорный компьютер

В любом учебнике написано, что компьютер состоит из процессора, памяти (оперативной и внешней) и устройств ввода-вывода. Процессор и оперативную память часто называют **центральными устройствами**, поскольку именно они составляют основу компьютера.

В данной статье мы будем рассматривать только центральные устройства, так как именно их взаимодействие меняется при добавлении процессоров. Для всех остальных устройств, напротив, число процессоров не существенно, и, следовательно, в рамках данной статьи они не интересны.

Напомним еще, что **оперативная память (ОЗУ — оперативное запоминающее устройство)** состоит из отдельных ячеек, каждая из которых имеет свой номер, называемый **адресом**.

В современных компьютерах процессор и память (а также все остальные устройства, которые мы договорились не рассматривать) обмениваются данными через **шину** (или **магистраль**). Начиная с самого первого школьного учебника информатики [7] функциональная схема взаимодействия изображается примерно так, как на рисунке 1.



Рис. 1. Однопроцессорный компьютер

Если не вдаваться в детали, то чтение в такой архитектуре выполняется следующим образом: процессор выставляет на шине требуемый адрес и делает запрос на чтение; память принимает запрос, находит нужные данные и по шине возвращает их процессору.

1.3. Кэш-память

Существует еще одна «тайная» (от *англ.* cache — тайник) разновидность памяти, которой нет на нашей схеме. И справедливо нет, ибо она не относится к архитектуре. Тем не менее говорить о ней при обсуждении многопроцессорной системы придется, так что кратко познакомимся с **кэш-памятью**.

Начнем с того, что кэш-память располагается между процессором и оперативной памятью. Это «технологическая» память, она нужна исключительно для ускорения процесса, а значит, согласно П2, к архитектуре не относится. Правило П1 для нее тоже справедливо, потому что ее работой аппаратно руководит процессор, а команды программы к данным кэш-памяти обращаться не могут [8].

Почему возникла потребность в кэш-памяти? Самый общий ответ звучит так: потому что память работает много медленнее процессора и тот вынужден долго ждать нужных ему данных. Проблема здесь в том, что инженеры не могут сделать память одновременно и большой, и быстрой. Или — или. Как невозможно добиться, чтобы грузовик одновременно перевозил много груза и при этом ехал быстро. Поэтому был придуман компромиссный вариант: небольшую, но быструю память ставят между процессором и остальной (большой и медленной) памятью. Именно в этой памяти хранятся наиболее часто используемые данные.

Идеи работы кэш-памяти объясняются на рисунке 2.

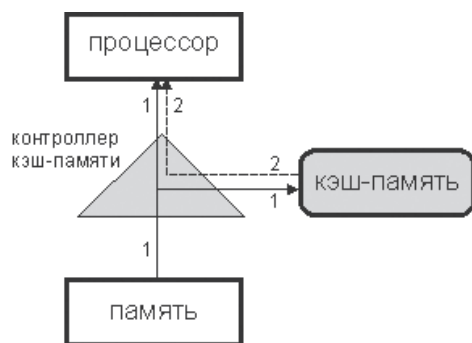


Рис. 2. Принцип работы кэш-памяти

Процессор общается с кэш-памятью через контроллер, запрашивая у него содержимое нужного адреса памяти. Первоначально кэш-память не содержит данных, поэтому при первом обращении контроллер вынужден извлекать содержимое непосредственно из оперативной памяти (ОЗУ). Прочитав значение, контроллер передает его процессору и при этом, что очень важно, оставляет в «подведомственной» кэш-памяти. При повторном обращении к этому адресу данные уже будут извлекаться не из ОЗУ, а из более быстрой кэш-памяти.

Важнейшая проблема в рассматриваемой организации памяти заключается в том, что процессор не только читает исходные данные из ОЗУ, но и записывает туда результаты. При этом приходится специально заботиться о соответствии содержимого кэша и ОЗУ. Обсуждение того, как это делается, выходит далеко за рамки нашего рассмотрения.

На самом деле картина еще сложнее, но общий принцип именно такой.

2. Многопроцессорная архитектура

Переходим, наконец, к самой интересной части. Как надо усовершенствовать схему рисунка 1, чтобы подключить к ней еще несколько процессоров?

2.1. Архитектура с общей памятью

Самое простое и естественное, что приходит в голову, это просто присоединить дополнительные процессоры к общей шине (рис. 3).



Рис. 3. Многопроцессорный компьютер с общей памятью

Такая схема имеет два достоинства: во-первых, она проста, а во-вторых, любые изменения данных, сделанные одним процессором, немедленно «видны» всем остальным, поскольку память общая. При не слишком большом количестве процессоров схема с общей памятью, действительно, будет работать. Но она обладает существенным недостатком. Дело в том, что каждый процессор занимает шину при обращении к памяти. Это гораздо серьезнее, чем кажется на первый взгляд, поскольку команды программ для каждого процессора

тоже читаются по шине из (той же самой) памяти. В теории многоядерных устройств часто считают [9], что если имеется N процессоров, то из-за потерь при разделении памяти производительность системы возрастает не в N , а только в $\sqrt{N}$ раз. Кроме того, N обычно не превышает 32–64 [2], потому что с большей нагрузкой шина просто не справляется.

Описанная архитектура с общей памятью имеет несколько эквивалентных названий, каждое из которых подчеркивает ее отдельные характерные особенности. В отечественной литературе такие системы часто называют *мультимикропроцессорными*. И это, действительно, удачный термин: много процессоров, все остальные устройства общие. Два других названия связаны с аббревиатурой SMP. Первая расшифровка этой аббревиатуры — Shared Memory Processors, т. е. *процессоры с разделяемой памятью*. (Не попадитесь здесь на игре слов: разделяемая она потому, что все процессоры разделяют одну общую память, а не потому, что память разделена между процессорами на отдельные части!) Вторая версия расшифровки — Symmetric Multi Processors, т. е. *симметричные микропроцессоры*, что подчеркивает одинаковые возможности всех процессоров.

Теперь подумаем, можно ли как-то улучшить схему с общей памятью. Обратим внимание на то, что когда параллельно работают несколько процессоров, то нагружается не только шина, но и сама память. В самом деле, одна микросхема памяти не может одновременно ответить даже двум процессорам одновременно, только по очереди! Тогда давайте сделаем несколько модулей памяти — появится теоретическая возможность параллельного обращения нескольких процессоров к разным частям памяти. Просто добавить к рисунку 1 еще несколько блоков памяти не получится, поскольку шина-то все равно одна. Значит, придется придумать более сложное соединение (*коммутацию*). Простейшее решение приведено на рисунке 4, где схематически показано присоединение двух модулей памяти к двум процессорам.

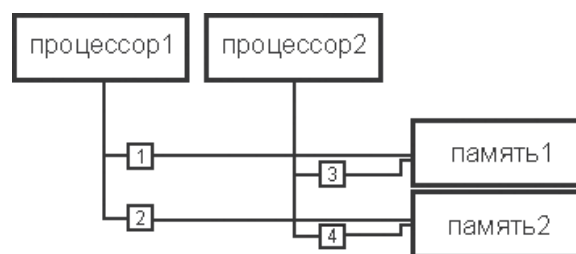


Рис. 4. Общая память из отдельных модулей

Из рисунка видно, что каждый процессор может подсоединиться к любому из блоков памяти через электронные переключатели 1–4. В некоторых случаях обмен может идти параллельно (например, через переключатели 1 и 4 или 2 и 3; вариант 1 и 3, как и 2 и 4, запрещен, ибо два одновременных обращения к одному модулю обслужить не удастся). Картина напоминает телефонную станцию, где абоненты попарно соединяются друг с другом. Главным недостатком такого матричного коммутатора является большой объем оборудования.

Реально используемых схем коммутации процессоров для систем с распределенной памятью намного больше [2].

2.2. Архитектура с распределенной памятью

Снова обратимся к рисунку 4. Подумаем, каким еще способом можно ускорить обмен между процессором и памятью. А что если к каждому процессору жестко присоединить собственный модуль памяти? С точки зрения производительности это, несомненно, будет эффективно, но зато в этом случае (вы заметили?) утратится механизм связи между процессорами через память. Как обычно, решив одну проблему, получили другую! Придется ввести иной механизм связи, соединив процессоры некоторым коммуникационным каналом. Теперь внимательно посмотрим на то, что получилось (рис. 5). На что похожа эта картинка?

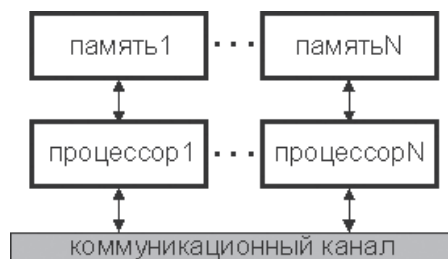


Рис. 5. Многопроцессорный компьютер с распределенной памятью

Совершенно верно, на независимые компьютеры, соединенные сетью. Не случайно системы с такой архитектурой называют **мультимпьютерными системами**. Более часто, правда, используется другой термин — **компьютеры с распределенной памятью**. Последнее название подчеркивает, что вместо общей памяти мы теперь получили распределенную между процессорами память.

Строго говоря, процессоры 1–N не обязательно должны быть одинаковыми, просто схема с идентичными процессорами практичнее.

Если взять несколько компьютеров, объединить их в сеть и заставить решать общую задачу, то тоже получится подобная вычислительная система. Часто ее называют **кластером**. С таких позиций Интернет можно считать огромнейшим параллельным компьютером с распределенной памятью.

2.3. Сравнение архитектур

Таким образом, из достаточно простых общих рассуждений нам удалось получить две различные параллельные архитектуры — с общей и с распределенной памятью (рис. 3 и 5). Такие два класса параллельных машин, действительно, существуют, и появились они не случайно. Они отражают два разных подхода к многопроцессорным системам: получить максимальную производительность (распределенная память) или упростить разработку программного обеспечения (общая память).

Заметим попутно, что двум различным архитектурам соответствуют разные технологии программирования:

- для архитектуры с общей памятью разработана **технология OpenMP** (Open Multi-Processing; <http://openmp.org>), которая основана на распределении данных в общей памяти между процессорами. Она позволяет очень легко распараллелить программу на одной машине;

- для архитектуры с распределенной памятью создана **технология MPI** (Message Passing Interface; <http://www.mpi-forum.org>), основанная на обмене сообщениями между процессорами. Чаше всего она используется на многомашинных кластерах.

Каждый класс параллельных компьютеров имеет свои достоинства и недостатки. «Для компьютеров с общей памятью проще создавать параллельные программы, но их максимальная производительность сильно ограничивается небольшим числом процессоров. А для компьютеров с распределенной памятью все наоборот» [2]. Нельзя ли объединить достоинства обоих классов? Одним из возможных путей является **архитектура NUMA** (Non Uniform Memory Access). Опуская детали, опишем ее суть. Каждый процессор имеет собственную память. Для обращения к памяти у каждого процессора есть специализированный контроллер, причем все контроллеры соединены общей шиной. Если адрес требуемых данных попадает в «собственную» (**локальную**) память процессора, то его содержимое считывается (или записывается) обычным образом. В противном случае контроллер отправляет запрос одному из своих «собратьев» с просьбой предоставить данные. В результате процессор, получивший значение от контроллера, даже не знает, из какой части памяти это значение пришло. Разумеется, имеется существенное различие в скорости получения данных: локальные ячейки читаются намного быстрее, чем удаленные (в два—семь раз [2]).

Трудности в технологии NUMA возникли с неожиданной стороны. Кэш-память, которая так эффективно ускоряла работу однопроцессорных машин (см. 1.3), принципиально «не подошла» для машин многопроцессорных. Суть проблемы заключается в следующем. Допустим, один процессор сохранил результат вычислений. Первоначально он попал в кэш, а в общей памяти пока по-прежнему лежит старое значение. Если другому процессору в этот момент потребуются значение рассматриваемой величины, то он получит (из памяти) неправильные данные. Конечно, и эта проблема решается. Разработана специальная модификация архитектуры — **ccNUMA** (cache coherent NUMA; *когерентность кэшей* означает их синхронизацию, согласование). Архитектура ccNUMA способна объединить 256 процессоров и более.

Несмотря на существование описанной компромиссной архитектуры, классическое деление на машины с общей и распределенной памятью по-прежнему остается в силе.

2.4. Кое-что о многоядерных микропроцессорах

По своей идее многоядерный микропроцессор — это реализация внутри одной микросхемы нескольких процессоров (попутно заметим, что в кристалле фиксированного размера не удастся вместо одного ядра разместить несколько, не уменьшая при этом их вычислительную мощность). Бесспорно, все микропроцессорные ядра попадут в одну машину, так что, казалось бы, ссылаемся на рисунок 3 и обсуждение закончено. Но не тут-то было. Дело в том, что неотъемлемой внутренней частью современного процессора стала кэш-память, которая даже имеет несколько уровней. Соединение разных уровней кэш-памяти существенно отличается у разных производителей. Читать техническую документацию дело очень

трудное, поэтому сошлюсь здесь на известную академическую книгу «Принципы параллельного программирования» [4]. Там проведен подробный сравнительный анализ двух двухъядерных процессоров — Intel Core Duo и AMD Dual Core Opteron. Подчеркнем, что процессоры данных ведущих производителей, несмотря на разное устройство, должны удовлетворять требованию совместимости по системе команд. Как оказывается, кэш второго уровня (L2) в первом процессоре общий, что, по сути дела, обеспечивает совместную обработку данных обоими ядрами. Во втором процессоре, напротив, все уровни кэш-памяти у каждого ядра свои. В результате требуются аппаратные средства синхронизации содержимого кэшей. С одной стороны, это усложняет схему, но с другой — существенно облегчается объединение нескольких микросхем для совместной работы.

Увеличение количества ядер не приводит к унификации (см., например, рис. 9.42 в книге [6]). Попутно заметим, что ядра процессоров могут быть разными, но мы не будем рассматривать такой случай.

Какие выводы для нас из этого следуют? В основном не очень обнадеживающие. Во-первых, внутреннее устройство многоядерных микропроцессоров сильно различается, во всяком случае, на данный момент. Во-вторых, работу многоядерного процессора не удастся объяснить, не привлекая детали строения кэш-памяти. По моему мнению, это очень неудачно для первого знакомства, поскольку, как мы видели, для однопроцессорного случая кэш частью архитектуры не является (и, следовательно, при начальном изучении можно его даже не рассматривать!). Заметим, что как минимум признак П1 остается в силе (проверить на практике П2, по-видимому, просто невозможно), так что по-прежнему остается в силе и вывод о непринадлежности кэша к архитектуре.

Таким образом, хотя процессоры с небольшим числом одинаковых ядер определенно тяготеют к классической архитектуре с общей памятью, у них есть некоторая специфика. Поэтому с безоговорочным приложением к ним классических схем мне бы пока хотелось повременить. Будущее покажет.

2.5. Еще о роли кэш-памяти

Этот раздел является дополнительным, так что при первом знакомстве он без особого ущерба может быть пропущен.

Для иллюстрации полезности понимания идей работы кэша обратимся к примеру из книги [4]. Там в первой главе подробно изучаются особенности параллельного программирования для простой задачи о подсчете в одномерном массиве количества элементов, которые удовлетворяют определенному условию. (Вполне школьная задача, не правда ли?) Нам сейчас интересен только один аспект решения: согласно параллельному алгоритму, сначала каждый процессор считает количество подходящих элементов в своей части массива, а затем все результаты складываются. Счетчики расположены в едином массиве. Оказалось, что такая параллельная программа выполняется даже дольше, чем обычная однопроцессорная, несмотря на то что все тонкости оптимизации под параллельную архитектуру ранее уже были выполнены. Причины плохой производительности нетривиальны и заключаются именно в особенностях работы кэш-памяти. Кратко рассмотрим проблему.

В кэш-памяти прочитанные значения чисел сохраняются не по одному, а строками; в строке может быть несколько значений. Такой способ связан с организацией чтения данных из памяти: компьютер всегда стремится прочитать за один раз как можно больше байт. Определенная логика в таком решении есть: если вы читаете элемент массива $C[0]$, то почти наверняка вам скоро понадобится $C[1]$, так что лучше заранее прочитать несколько соседних чисел. Однако в обсуждаемой параллельной программе такая «предусмотрительность» кэш-памяти сыграет с программистом злую шутку, и вот почему.

Будем для простоты рассматривать работу только двух процессоров — P_0 и P_1 . Первый использует счетчик $C[0]$, а второй — $C[1]$. Предположим, что оба счетчика при чтении попадают в кэш-памяти у каждого процессора в одну строку. И вот тут начинается самое неожиданное. Допустим, что P_0 увеличил значение счетчика $C[0]$, изменив тем самым часть строки в своей кэш-памяти. «Мудрые» средства синхронизации содержимого кэшей тут же «доложили» в P_1 , что строка, содержащая $C[0]$ (и не забудьте, $C[1]$ тоже), больше недействительна и ее придется заменить. Нам с вами известно, что P_0 не трогал переменную $C[1]$, но P_1 -то об этом не знает! В итоге вся строка в кэш P_1 обновляется, хотя, по сути дела, зря. Но и это еще не все неприятности. P_1 тоже может увеличить свой счетчик на единицу. Тем самым он ответно осложнит жизнь P_0 , заставив его тоже обновить свой кэш, хотя, как вы понимаете, тоже напрасно. Просто счетчикам не повезло, что они оба попадают в одну строку кэш-памяти. Таким образом, P_0 и P_1 все время мешают друг другу, заставляя излишне часто обновлять содержимое кэш-памяти. Подобное негативное взаимовлияние даже получило специальное название — *ложное совместное использование*.

Конечно, вскрыв причину, авторы книги [4] немедленно устранили ее и показали, что только тогда программа действительно дала выигрыш, близкий к ожидаемому.

О чем говорит нам эта история? Она показывает, что уже для написания простой учебной параллельной программы нужны знания об аппаратной части. Причем порой даже представлений на уровне архитектуры может оказаться недостаточно. Казалось бы, зачем тем, кто работает с языком высокого уровня (по определению машинно-независимым), знать особенности работы аппаратной части? А вот в рассмотренном случае без этого вообще не удалось бы получить ускоряющую вычисления параллельную программу... «Все механизмы кэширования в основном прозрачны для прикладных программ и после разрешения кэширования пропускают через себя потоки инструкций и данных без требования явного программного управления. Однако знание особенностей механизмов кэширования помогает в оптимизации кода» [3].

3. Учебное программное обеспечение

Для того чтобы можно было закрепить изложенный выше теоретический материал на практических занятиях, были разработаны две программы, демонстрирующие работу основных параллельных архитектур. Они не являются универсальными и обе решают одну и ту же типовую задачу.

3.1. Постановка задачи

Рассмотрим обработку элементов двумерного массива из N строк по N элементов каждая. Что именно делается с элементами, большого значения не имеет, важно только, чтобы время обработки каждой строки было одинаковым. Обозначим это время через T_s . Тогда полное время вычислений на обычном однопроцессорном компьютере легко определить:

$$T1 = N \cdot T_s. \quad (1)$$

Рассмотрим теперь, как быстро будут получены решения той же задачи на многопроцессорной вычислительной системе каждого из двух базовых типов.

3.2. Система с общей памятью

Пусть теперь у нас есть многопроцессорная машина с общей памятью и в ней C процессоров (или ядер). Теперь обработка N строк будет разделена между процессорами, так что на каждый из них придется по N/C строк (здесь и далее везде считаем, что в результате деления получится целое число строк).

Из теоретического раздела 2.1 мы знаем, что при подсоединении к шине нескольких процессоров доступ к памяти замедляется. Если обозначить связанный с этим коэффициент через $K1$, то время обработки одной строки увеличится до $K1 \cdot T_s$. В итоге полное время вычислений на параллельной машине будет равно:

$$Tc = \frac{N}{C} \cdot K1 \cdot T_s = \frac{K1}{C} \cdot T1. \quad (2)$$

Уменьшение времени счета S получим, поделив $T1$ на (2):

$$S = \frac{T1}{Tc} = \frac{C}{K1}.$$

Наконец, полагая $K1 = \sqrt{C}$, имеем простую окончательную формулу:

$$S = \sqrt{C}. \quad (3)$$

Для моделирования процесса решения задачи на однопроцессорной и многопроцессорной машине с общей памятью была разработана и реализована специальная программа-имитатор*. Вид ее основного окна приведен на рисунке 6.

Левая панель изображает машину с одним процессором, а правая — с несколькими (с помощью радиокнопок их число можно задать равным 2, 4 или 8). Рядом с процессором в обеих машинах схематически изображен участок памяти, в котором хранится обрабатываемый массив 8×8 . Подчеркнем, что реальная память компьютера, конечно, одномерна (вспомните, как вводится понятие адресов ячеек), но использование при визуализации двумерного массива значительно нагляднее для учеников.

Знаком «#» помечаются элементы массива, которые обработал обычный процессор. В случае нескольких процессоров элемент помечается номером того процессора, который его обработал. На рисунке 6 изображен случай с восемью процессорами, так что каждый производил вычисления в собственной строке массива. Видно также, что когда параллельная машина выполнила 100 % работы, индикатор обычного компьютера замер на отметке 35 %.

Чтобы увидеть более детальные количественные результаты, нажмем кнопку *Анализ*. Появится новое окно (рис. 7).

Здесь наиболее интересная часть — это диаграмма производительности в правой части окна. Левый белый столбик соответствует обычному процессору, его высота принята за 1. Средний, самый высокий столбик характеризует наши ожидания: поскольку у нас восемь процессоров, то нам хотелось бы увеличения производительности в восемь раз. Но в действительности результат меньше (столбик крайний справа) — реальное ускорение будет всего в 2,8 раза.

* Программы-имитаторы, описанные в статье, можно скачать на сайте ИНФО: http://infojournal.ru/journals/school/school_02-2016/

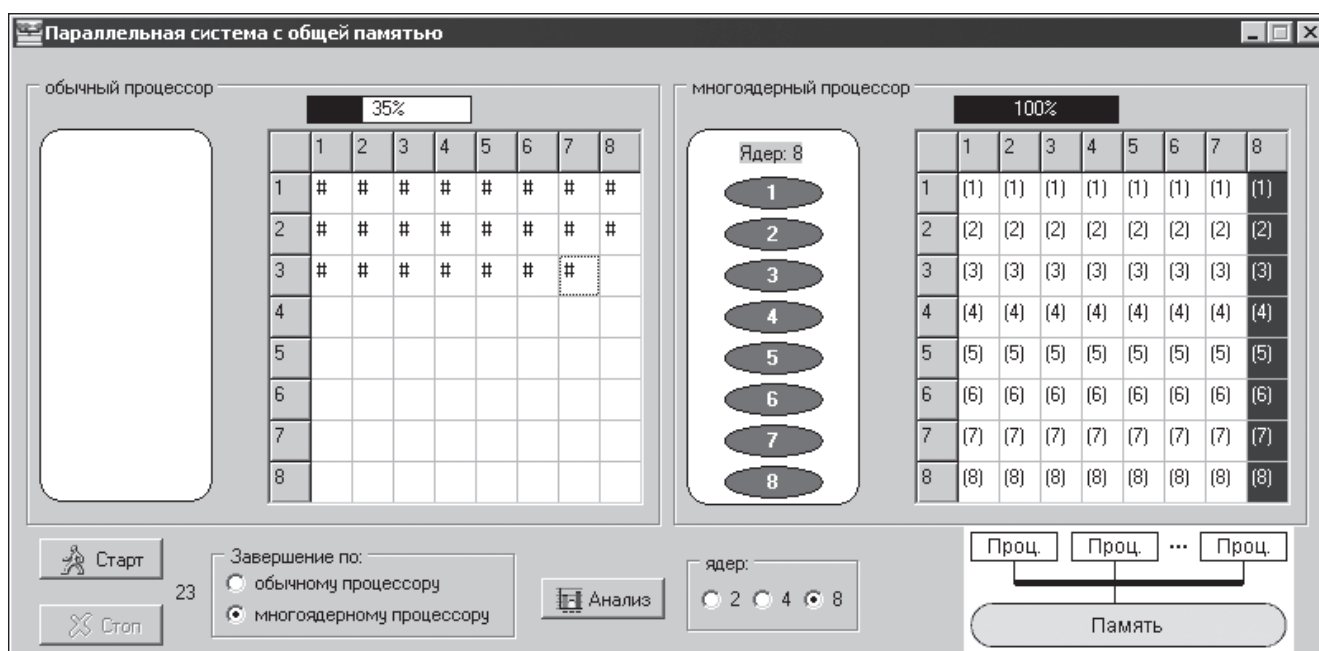


Рис. 6. Программа-имитатор параллельной системы с общей памятью

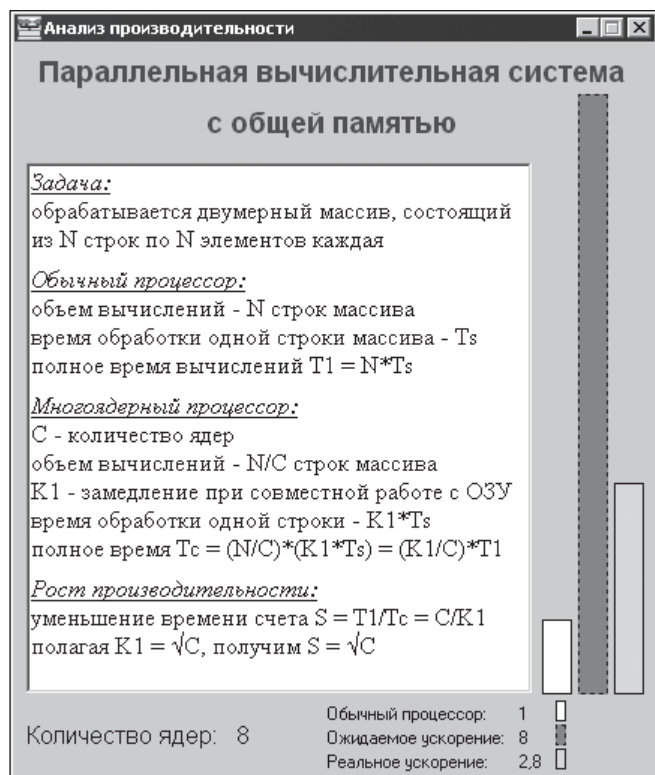


Рис. 7. Окно с анализом результатов

Можно попробовать выбрать четыре или два процессора — результаты будут еще более скромными.

Обратим внимание читателей на то, что процесс обработки массива специально визуализируется в замедленном темпе так, чтобы можно было рассмотреть детали.

3.3. Система с распределенной памятью

Рассмотрим теперь вычислительную систему с распределенной памятью. Пусть в ней P процессоров, у каждого из которых собственная память. Будем полагать, что начальные значения массива хранятся в памяти одного из процессоров и он передает каждому из остальных процессоров его долю чисел. Одну часть он оставляет себе и, завершив распределение чисел, обрабатывает ее сам. Таким образом, главным отличием от случая с общей памятью является необходимость в предварительном копировании данных. Очевидно, что это будет замедлять вычисления, снижая выигрыш от параллельной обработки.

Оценим величину полученного выигрыша производительности.

Каждый процессор обработает N/P строк массива. Работая с собственной памятью, он вычислит результат для каждой строки так же быстро, как это делает обычный процессор, — за время T_s (см. 3.1). Таким образом, общее время обработки будет равно $N \cdot T_s / P$. Подчеркнем, что это не только время работы *одного* процессора, но и *общее* время, так как все процессоры работают параллельно. К сожалению, не все так здорово: есть еще копирование строк. Обозначим время копирования одной строки T_{copy} , тогда передача данных одному процессору займет время $N \cdot T_{copy} / P$. Всего таких копирований будет $P - 1$ (последнюю часть, как вы помните, трогать не надо!). В итоге полное копирование потребует:

$$(P - 1) \cdot \frac{N \cdot T_{copy}}{P}.$$

Сложим время копирования и время счета:

$$T_p = \frac{N}{P} [(P - 1) \cdot T_{copy} + T_s].$$

Вынесем T_s за скобку, и, учитывая (1), получим:

$$T_p = \frac{(P - 1)K_2 + 1}{P} T_1,$$

где $K_2 = \frac{T_{copy}}{T_s}$ есть важный параметр задачи. Он характеризует конкуренцию двух факторов: скорости обработки строки T_s и потерь на ее подготовку T_{copy} . Обратите внимание, что результат зависит не от времени копирования и не от времени обработки по отдельности, а от их соотношения!

Наконец, сокращая T_1 , получим окончательную формулу для повышения производительности:

$$S = \frac{T_1}{T_p} = \frac{P}{(P - 1)K_2 + 1}. \quad (4)$$

Таким образом, ускорение зависит от числа процессоров и от величины K_2 , которая, напомним, есть отношение времени копирования и обработки строки.

Из формулы (4) уже можно увидеть некоторые важные закономерности.

При $K_2 = 1$ (строка копируется столько же, сколько считается) формула дает $S = 1$, т. е. никакого выигрыша от многопроцессорности нет.

При стремлении K_2 к 0 (данные копируются очень быстро) результат стремится к P , т. е. к максимально возможному (проверьте, подставляя 1 и 0 вместо K_2 в (4)).

При $K_2 < 1$, когда время копирования меньше времени обработки, ускорение от параллельного выполнения всегда будет (проверьте, например, подстановкой конкретных дробных числовых значений).

Зато при $K_2 > 1$, когда строки копируются долго, результат будет даже *хуже*, чем на обычном компьютере! Как видите, потери вполне могут «съесть» весь выигрыш от параллельной обработки.

Чтобы исследовать поведение параллельной системы с распределенной памятью, запустите вторую программу-имитатор (рис. 8).

Работа с этой программой во многом похожа на описанную в предыдущем разделе. Поэтому остановимся только на особенностях.

Процессор, который расположен в нижней части окна (в нашем случае он имеет номер 4), всегда считается главным: в его памяти хранится исходный массив. Он «раздает» строки процессорам 1–3. Из рисунка видно, что процессор 1 обрабатывает первые две строки, процессор 2 — третью и четвертую и т. д. Две последние строки не копируются, и процессор 4 занимается ими сам.

Отмечу еще одну интересную, на мой взгляд, особенность: алгоритм для симметричной системы процессоров не удастся сделать симметричным! Появляется выделенный процессор, распределяющий данные.

С помощью радиокнопок можно задать три фиксированных значения для K_2 . Рядом расположен флажок, которым можно отключить процесс копирования. В последнем случае имитатор «таинственным образом» заранее распределит строки массива между процессорами (не спрашивайте, как!) Опробуйте этот вариант,

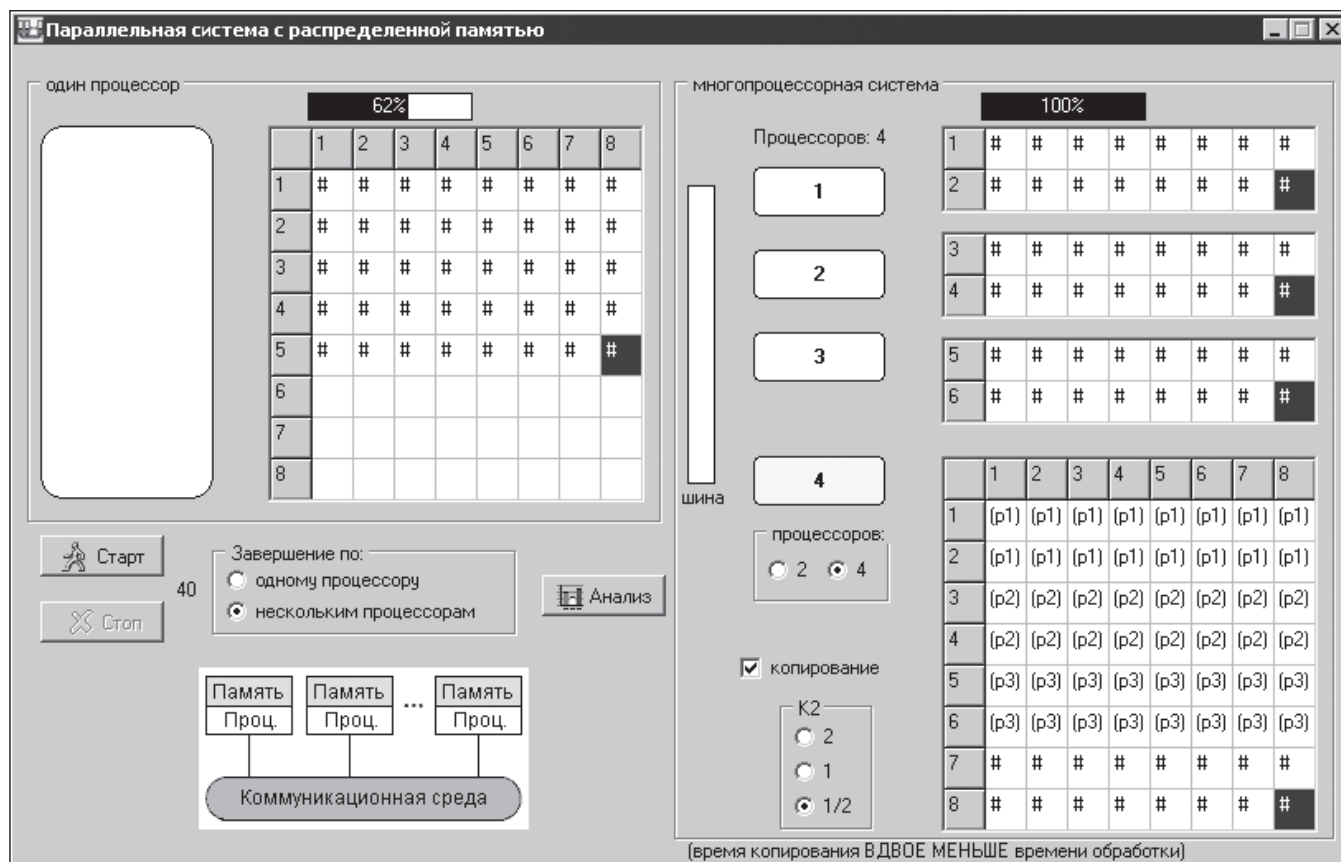


Рис. 8. Программа-имитатор параллельной системы с распределенной памятью

и вы с радостью увидите, что при четырех процессорах результат, действительно, будет получен в четыре раза быстрее! Беда в том, что в жизни так не сделать.

3.4. О работе с программами-имитаторами

Программы-имитаторы подробно визуализируют процесс параллельной обработки массива. Некоторая проблема состоит в том, что при работе с ними требуется внимательно смотреть и анализировать то, что видишь (к сожалению, не всем современным ученикам это удастся с одинаковым успехом).

В связи с этим можно порекомендовать следующий порядок знакомства с каждой из программ:

1. Запустить программу и внимательно изучить ее окно. При этом обратить внимание на две категории деталей: а) органы визуализации однопроцессорного и многопроцессорного компьютеров (процессоры и память); б) органы управления программой.
2. Нажать кнопку *Старт* и понаблюдать за изменениями на экране. Выяснить, как отмечается обработка элемента массива, чем отличается работающий процессор от простаивающего и т. д. Возможно, поменять значения параметров, пока не особенно обращая внимание на время выполнения. Постараться увидеть за каждым изменением на экране некоторый этап решения задачи.
3. Обратить внимание, есть ли возможность одновременного выполнения тех или иных действий, например, копирования данных в один процессор и их обработки в другом.

4. Попрактиковаться в визуальном сравнении скорости обработки информации в левой и правой панелях: понаблюдать за перемещением индикаторов, стараясь заметить, какой идет быстрее; сопоставить количество обработанных элементов за единицу времени.
5. Нажать кнопку *Анализ* и внимательно изучить информацию в появившемся окне.
6. Наконец, когда уже есть понимание правил визуализации, можно перейти к анализу влияния параметров задачи (например, числа процессоров) на ускорение параллельного алгоритма. Лучше всего занести полученные результаты в таблицу. Проанализировать получившиеся числа; по заданию учителя сравнить их с результатами вычисления по формулам.
7. Сделать выводы по проделанной работе. Очень полезно сравнить между собой полученные для обеих архитектур результаты.

Заключение

Предлагаемая методика позволяет наглядно продемонстрировать следующие **идеи и принципы параллельных вычислений**:

- Многопроцессорная архитектура есть естественное развитие архитектуры классической, происходящее из-за стремления повысить производительность вычислительной системы.
- Существуют два основных класса организации параллельных вычислительных систем — с общей и с распределенной памятью.

- В первом из названных классов обмен данными между процессорами осуществляется за счет общей памяти. Как следствие при этом нет необходимости в специальной передаче данных от одного процессора другому.
- Архитектура с общей памятью не позволяет объединить много процессоров. За счет возрастания нагрузки при многократных обращениях к памяти эффективность от параллельной работы процессоров падает.
- Разделение памяти на отдельные блоки улучшает условия для параллельной обработки данных.
- Архитектура с общей памятью легче программируется.
- Архитектура с распределенной памятью по своей сущности представляет собой объединение самостоятельных компьютеров. Эта задача решается легче и позволяет объединять большее количество процессоров.
- При обработке в системе с распределенной памятью требуется передавать данные от одного процессора другому. Это снижает выигрыш от параллельности обработки.
- Архитектура с распределенной памятью сложнее программируется.
- Каждой из двух архитектур соответствует своя технология программирования.
- По многим причинам в системе с P процессорами не удается на практике достичь выигрыша в P раз. (Отметим, что в статье рассмотрены далеко не все причины такого положения.)

Проведенный анализ также позволил выделить следующие **методические аспекты преподавания основ параллельной архитектуры**:

- При отборе материала для изучения полезно ориентироваться именно на архитектуру, которая по определению включает в себя наиболее фундаментальные принципы устройства компьютера.

- По мнению автора, при изложении темы целесообразно базировать ее на сведениях о классической однопроцессорной архитектуре, а не излагать как независимый материал.
- С одной стороны, кэш-память создана как вспомогательное устройство, с другой — ее роль в современных процессорах очень велика. При этом механизмы работы кэша достаточно сложны. В связи с этим существует методическая проблема отбора необходимого для описания работы кэша учебного материала.
- Многоядерные процессоры имеют определенную специфику по сравнению с классической многопроцессорной архитектурой. Кроме того, их устройство в настоящее время бурно совершенствуется. Это еще одна проблема при объяснении темы.

Литература

1. Босова Л. Л. Параллельные алгоритмы в начальной и основной школе // Информатика в школе. 2015. № 2.
2. Воеводин В. В., Воеводин Вл. В. Параллельные вычисления. СПб.: БХВ-Петербург, 2002.
3. Лук М. Аппаратные средства IBM PC. Энциклопедия. СПб.: Питер, 2003.
4. Лин К., Снайдер Л. Принципы параллельного программирования. М.: Изд-во Московского университета, 2013.
5. Немнюгин С. А., Стесик О. Л. Параллельное программирование для многопроцессорных вычислительных систем. СПб.: БХВ-Петербург, 2002.
6. Орлов С. А., Цилькер Б. Я. Организация ЭВМ и систем. СПб.: Питер, 2011.
7. Основы информатики и вычислительной техники: проб. учеб. пособие для сред. учеб. заведений. В 2 ч. Ч. 2 / А. П. Ершов, В. М. Монахов, А. А. Кузнецов и др.; под ред. А. П. Ершова, В. М. Монахова. М.: Просвещение, 1986.
8. Столинс В. Структурная организация и архитектура компьютерных систем. М.: Вильямс, 2002.
9. Hill M. D., Marty M. R. Amdahl's Law in the Multicore Era // IEEE Computer. 2008. Vol. 41. № 7.

НОВОСТИ

Школьный шифр

В Москве, в здании президиума РАН, наградили победителей и призеров юбилейной XXV Межрегиональной олимпиады школьников по математике и криптографии.

Ее организаторы — Академия криптографии РФ, Академия ФСБ России и федеральное учебно-методическое объединение «Информационная безопасность». Поддержку олимпиаде традиционно оказывает и общественный совет при ФСБ РФ.

Для большинства школьников криптография остается загадочной наукой, хотя многие незаметно для себя ежедневно пользуются ее достижениями и технологиями. При этом многие ученые относят криптографию к числу основных источников новых задач и целых научных направлений в математике.

— Олимпиады школьников по математике и криптографии проводятся в нашей стране уже 25 лет. За эти годы многие школьники, участвовавшие в них, получили высшее образование и успешно работают в избранной

профессии, — заявил перед награждением победителей вице-президент Академии криптографии РФ Владимир Сачков.

В этом году свои силы в действительно сложном испытании попробовало рекордное количество школьников VIII—XI классов — около 4000 ребят. Ее очный тур проводился в 44 городах России, и в этом году он впервые прошел в Севастополе. Случайно или нет, но именно в этот день в Крыму полностью отключили свет из-за энергетической блокады, и, чтобы не сорвать решение задания, организаторам пришлось срочно искать генератор, чтобы зарядить ноутбуки. О сложности криптографических задач говорит тот факт, что из всех 4000 участников жюри в итоге отметило дипломами призеров только 231 человека. Из них 86 — это ученики XI класса.

Многие из ребят, как признаются, собираются поступать в профильные вузы и в будущем стать если не профессиональными криптографами, то как минимум специалистами по компьютерной и информационной безопасности.

(По материалам «Российской газеты»)