Курс, как вы уже поняли, называется разработкой параллельных и распределенных программ, и будет поделен на две логические части, соответствующие его названию – параллельные и распределенные вычисления. Вопрос того, чем они между собой отличаются, возникает часто и между собой эти вещи путаются и считаются синонимами.

Поэтому перед тем, как начать изучение различных видов разветвления вычислительных процессов, необходимо определить базовые понятия. Давайте разберемся, чем они отличаются.

Распределенные вычисления — способ решения трудоёмких вычислительных задач с использованием нескольких компьютеров, объединённых в параллельную вычислительную систему.

Параллельные вычисления — такой способ организации компьютерных вычислений, при котором программы разрабатываются как набор взаимодействующих вычислительных процессов, работающих параллельно.

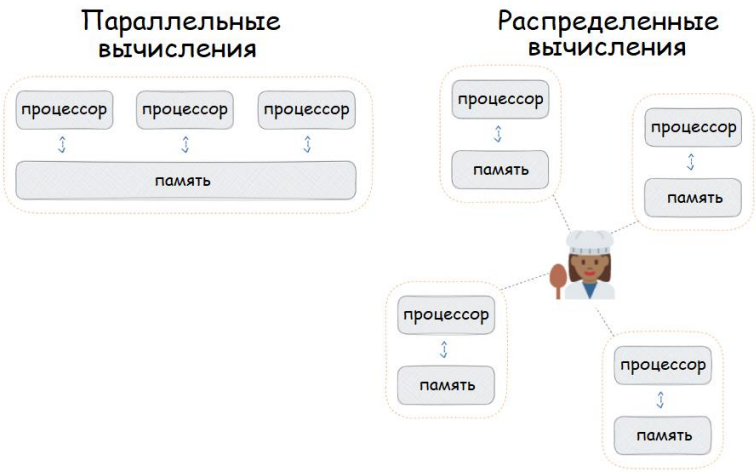
Таким образом распределенные вычисления не могут производится на одной вычислительной машине, а параллельные вычисления могут производиться как на одной (многопоточность), так и на нескольких машинах. Возможность выполнения параллельных вычислений в распределенных системах и приводит к частому заблуждению, что это одно и тоже.

Распределенная система — это набор независимых компьютеров, представляющиеся их пользователям единой объединенной системой.

Параллельные вычисления — вычисления, которые можно реализовать на многопроцессорных системах с использованием возможности одновременного выполнения многих действий, порождаемых процессом решения одной или многих задач.

Ещё одно отличие состоит в том, каким образом в вычислениях используется память. Параллельные вычисления выполняют задачи, используя множество процессоров, разделяющих одну и ту же память. Эта разделяемая память необходима, поскольку отдельные процессы работают вместе над одной и той же задачей. Параллельные вычислительные системы ограничены количеством процессоров, способных работать с разделяемой памятью.

С другой стороны, распределенные вычисления выполняют задачи на разных компьютерах, не имеющих общей памяти; эти компьютеры общаются друг с другом путем передачи сообщений. Это означает, что общая задача разбивается на несколько отдельных задач, центральный планировщик объединяет результаты всех индивидуальных задач и возвращает общий результат пользователю. Распределенные вычислительные системы теоретически можно расширять до бесконечности.




Разобравшись с этим существенным вопросом, переходим к введению в первый модуль, если так можно сказать – параллельное программирование.

В активе человечества имеется не так много изобретений, которые, едва возникнув, быстро распространяются по всему миру. Одним из них является, конечно, персональный компьютер. Появившись в середине двадцатого столетия, он через несколько десятков лет стал повсеместным и незаменимым инструментом для обработки самой разнообразной информации: цифровой, текстовой, визуальной, звуковой и др.

Известно, что первопричиной создания компьютера была настоятельная необходимость быстрого проведения вычислительных работ, связанных с решением больших научно-технических задач: в атомной физике, авиа- и ракетостроении, климатологии и др. Решение таких задач и сейчас остаётся главным стимулом совершенствования компьютеров, однако в настоящее время распространены и сферы применения, в которых явная вычислительная составляющая либо отсутствует, либо занимает лишь малую и незначительную часть.

Работа биржи, управление производством, офисные приложения, информационные системы, системы образования, видеоигры, ведение хозяйств – лишь отдельным примеры областей невычислительного использования компьютеров. Я отметил про явную вычислительную составляющую именно с тем расчётом, что на самом деле во всех этих областях всё равно так или иначе присутствуют вычисления – та же видеоигровая графика очень серьезно базируется именно на них, но в итоге конечной целью пользователя не является получение результатов каких-либо расчётов. В подобных областях вполне приемлемо использование ПК и рабочих станций для достижения тех или иных целей. Относительная простота процесса использования и комфортная программная среда формируют устойчивое впечатление о компьютере как о хорошем помощнике. И мало кто из рядовых пользователей догадывается и тем более знает, что при переходе к решению крупных и очень крупных задач всё радикально меняется. Сами ПК становятся очень сложными, исчезает дружественность интерфейса, программная среда переходит на командный язык и начинает требовать от пользователей предоставления не всегда известной информации.

Сами по себе большие задачи прямо сейчас не является предметом детального обсуждения курса, к этому мы подберемся, когда будем изучать распределенные вычисления. Иногда к ним всё равно будет произведено явное обращение для того, чтобы проиллюстрировать то или иное положение. Неявно любое обсуждение чаще всего предполагает, что решается скорее всего именно большая задача.

Все, что связано с большими компьютерами и большими задачами, сопровождается характерным словом "параллельный": параллельные компьютеры, параллельные вычислительные системы, языки параллельного программирования, параллельные численные методы и т. п. В широкое употребление этот термин вошел почти сразу после появления первых компьютеров. Точнее, почти сразу после осознания того факта, что созданные компьютеры не в состоянии решить за приемлемое время многие задачи. Выход из создавшегося положения напрашивался сам собой. Если один компьютер не справляется с решением задачи за нужное время, то попробуем взять два, три, десять компьютеров и заставим их *одновременно* работать над различными частями общей задачи, надеясь получить соответствующее ускорение. Идея показалась плодотворной, и в научных исследованиях конкретное число объединяемых компьютеров довольно быстро превратилось в произвольное и даже сколь угодно большое число.

Объединение компьютеров в единую систему потянуло за собой множество следствий. Чтобы обеспечить отдельные компьютеры работой, необходимо исходную задачу разделить на фрагменты, которые можно выполнять *независимо* друг от друга. Так стали возникать специальные численные методы, допускающие возможность подобного разделения. Чтобы описать возможность одновременного выполнения разных фрагментов задачи на разных компьютерах, потребовались специальные языки программирования, специальные операционные системы и т. д. Постепенно такие слова, как "одновременный", "независимый" и похожие на них стали заменяться одним словом *"параллельный"*. Всё это синонимы, если иметь в виду описание каких-то процессов, действий, фактов, состояний, не связанных друг с другом. Ничего иного слова "параллелизм" и "параллельный" в областях, относящихся к компьютерам, не означают.

Далеко не сразу удалось объединить большое число компьютеров. Первые компьютеры в принципе были слишком громоздкими и потребляли слишком много энергии. Но со временем успехи микроэлектроники привели к тому, что важнейшие элементы компьютеров по многим своим параметрам, включая упомянутые размер и объем потребляемой энергии, стали в тысячи раз меньше, поэтому объединение большого числа компьютеров и их вычислительных ядер в единую систему стала главенствовать в повышении общей производительности вычислительной техники. В суперкомпьютерах достигаются весьма впечатляющие суммарные характеристики. Можно взглянуть на то, как они эволюционировали за последние 20 лет: суперкомпьютер ASCI White компании IBM, впервые запущенный в 2001 году, имел 512 машин по 16 процессоров каждая (суммарно 8192 процессора, работающих на частоте 375 МГц), 6 ТБ ОЗУ и 160 ТБ дискового массива; пиковая производительность составляла более 12 терафлопс ($12*10^{12}$ флопс). В то же время один из самых современных суперкомпьютеров Frontier, или OLCF-5, спонсированный Hewlett-Packard Enterprise (HP) и имеющий свыше 8 млн вычислительных ядер засчёт вычислительных и графических процессоров AMD, имеет пиковую производительность 1.6 эксафлопс ($1.6*10^{18}$ флопс). Флопс (от англ. FLOPS – FLoating point OPerations per Second) – внесистемная единица для измерения производительности

компьютеров, показывающая количество операций над числами с плавающей запятой, которое данная система может выполнять за секунду времени. Для сравнения масштаба суперкомпьютеров и обычных процессоров ПК, Intel Core i9-9900k имеет пиковую производительность в 460 гигафлопс. Подробнее о том, что вообще из себя представляет пиковая производительность в чем её отличие от реальной, будет рассказано в следующих лекциях. Как бы то ни было, данная демонстрация позволяет показать, насколько возросла производительность за годы, хоть и стоит понимать, что это лишь пиковая производительность и для приближения к её достижению на уровне реальных вычислительных задач необходимо правильно уметь воспользоваться предоставленными мощностями и оборудованием.

Параллелизм на различных уровнях характерен для всех современных компьютеров от персональных до супербольших: одновременно функционирует множество процессоров, передаются данные по коммуникационной сети, работают устройства ввода/вывода, осуществляются другие действия. Любой параллелизм направлен на повышение эффективности работы компьютера. Некоторые его виды реализованы жестко в "железе" или обслуживающих программах и недоступны для воздействия на него рядовому пользователю. Но с помощью жесткой реализации не удается достичь наибольшей эффективности в большинстве случаев. Поэтому многие виды параллелизма реализуются в компьютерах гибко, и пользователю предоставляется возможность распоряжаться ими по собственному усмотрению.

Под термином "параллельные вычисления" как раз и понимается вся совокупность вопросов, относящихся к созданию ресурсов параллелизма в процессах решения задач и гибкому управлению реализацией этого параллелизма с целью достижения наибольшей эффективности использования вычислительной техники.

Вот уже более полувека параллельные вычисления привлекают внимание самых разных специалистов. Три обстоятельства поддерживают к ним постоянный интерес. Во-первых, это очень важная сфера деятельности. Занимаясь параллельными вычислениями, исследователь понимает, что он делает что-то, относящееся к самым большим задачам, самым большим компьютерам и, следовательно, находящееся на передовом фронте науки. Как минимум, близость к передовому фронту науки вдохновляет. Во-вторых, это очень обширная сфера деятельности. Она затрагивает разработку численных методов, изучение структурных свойств алгоритмов, создание новых языков программирования и многое другое, связанное с интерфейсом между пользователем и собственно компьютером. Параллельные вычисления тесно связаны и с самим процессом конструирования вычислительной техники. Структура алгоритмов подсказывает необходимость внесения в компьютер изменений, эффективно поддерживающих реализацию структур-

ных особенностей. Инженерные же новшества стимулируют разработку новых алгоритмов, эффективно эти новшества использующих. И, наконец, в-третьих. С формальных позиций рассматриваемая сфера деятельности легко доступна для исследований. Достаточно более или менее познакомиться с ее основами на уровне популярной литературы и уже можно делать содержательные выводы, возможно, даже никем не опубликованные.

С какой бы стороны не рассматривать параллельную вычислительную технику, главным стимулом ее развития было и остается повышение эффективности процессов решения больших и очень больших задач. Эффективность зависит от производительности компьютеров, размеров и структуры их памяти, пропускной способности каналов связи. Но в не меньшей, если не большей, степени она зависит также от уровня развития языков программирования, компиляторов, операционных систем, численных методов и многих сопутствующих математических исследований. Если с этой точки зрения взглянуть на приоритеты пользователей, то они всегда связаны с выбором тех средств, которые позволяют решать задачи более эффективно.

Эффективность — понятие многоплановое. Это удобство использования техники и программного обеспечения, наличие необходимого интерфейса, простота доступа и многое другое. Но главное — это достижение близкой к пиковой производительности компьютеров. Данный фактор настолько важен, что всю историю развития вычислительной техники и связанных с ней областей можно описать как историю погони за наивысшей эффективностью решения задач.

Конечно, такой взгляд отражает точку зрения пользователей. Но ведь именно пользователям приходится "выжимать" все возможное из имеющихся у них средств и приводить в действие все "рычаги", чтобы достичь максимальной производительности компьютеров на своих конкретных задачах. Поэтому им нужно знать, где находятся явные и скрытые возможности повышения производительности и как наилучшим образом ими воспользоваться. Реальная производительность сложным образом зависит от всех составляющих процесса решения задач. Можно иметь высокопроизводительный компьютер. Но если компилятор создает не эффективный код, реальная производительность будет малой. Она будет малой и в том случае, если не эффективны используемые алгоритмы. Не эффективно работающая программа — это прямые потери производительности компьютера, средств на его приобретение, усилий на освоение и т. п. Таких потерь хотелось бы избежать или, по крайней мере, их минимизировать.

Проблемы пользователей нам известны не понаслышке. За плечами лежит более двадцати лет научной, производственной и педагогической деятельности в области параллельных вычислений. За это время освоены многие компьютеры и большие вычислительные системы. Было решено немало задач. Но не было ни одного случая, когда одна и та же программа эффективно

реализовывалась без существенной переделки при переходе к другой технике. Конечно, хотя и трудно, но все же можно заново переписать программу, удовлетворяя требованиям языка программирования и штатного компилятора. Однако новую большую программу суперсложно сделать эффективно работающей.

Технология обнаружения узких мест процесса реализации программы плохо алгоритмизирована. Опыт показывает, что для повышения эффективности приходится рассматривать все этапы действий, начиная от постановки задачи и кончая изучением архитектуры компьютера, через выбор численного метода и алгоритма, тщательно учитывая особенности языка программирования и даже компилятора. Основной способ обнаружения узких мест — это метод проб и ошибок, сопровождающийся большим числом прогонов вариантов программы. Необходимость многих экспериментов объясняется скудностью информации, получаемой от компилятора после каждого прогона. К тому же ни один компилятор не дает никаких гарантий относительно меры эффективности реализуемых им отдельных конструкций языка программирования. С самого начала пользователь ставится в такие условия, когда он не знает, как надо писать эффективные программы. Получить соответствующую информацию он может только на основе опыта, изучая почти как черный ящик влияние различных форм описания алгоритмов на эффективность.

Исследования показывают, что большинство узких мест при разработке параллельного ПО можно объединить в три группы. Первая связана собственно с компьютером: на любой параллельной машине не все потоки данных обрабатываются одинаково. Какие-то реализуются эффективно, какие-то плохо. Важно понимать, что эти потоки представляют из себя при изучении особенностей архитектуры компьютера. Вторая группа – структура связей между операциями программы, поскольку не в каждой программе обязаны существовать фрагменты, эффективно реализуемые на конкретном компьютере. Третья группа – используемые в каждом конкретном компиляторе технологии отображения программ в машинный код.

Если технология позволяет разложить программу на фрагменты, по которым почти всегда будет строиться эффективный код, то узких мест в этой группе может не быть. Такие технологии были разработаны для первых параллельных компьютеров. Но по мере усложнения вычислительной техники технологии компиляции становятся все менее и менее эффективными, и узких мест в третьей группе оказывается все больше и больше. Для больших распределенных систем узкие места компиляции начинают играть решающую роль в потере общей эффективности. Поэтому уже давно наметилось и теперь почти воплотилось в реальность "компромиссное" разделение труда: наименее алгоритмизированные и сложно реализуемые этапы компиляции, в первую очередь, расщепление программы на параллельные ветви вычислений, распределение данных по модулям памяти и организацию пересылок данных возлагаются на пользователя. Проблем от этого не стало меньше. Просто о том, о чем раньше заботились разработчики компиляторов, теперь должны беспокоиться сами пользователи.

Очень важно акцентированное внимание к узким местам процесса решения задач. Узкие места описываются и изучаются практически всюду: в параллельных компьютерах и больших вычислительных системах, процессах работы многих функциональных устройств, конструировании численных методов, языках и системах программирования, различных приложениях и в работе пользователей. Особое внимание уделяется узким местам, связанным с выявлением параллельных структур алгоритмов и программ и их отображением на архитектуру компьютеров.

Мы уже говорили о том, что компьютер способен выполнять лишь программы, написанные в машинных командах. Однако программировать в машинных командах исключительно трудно. Прежде всего, из-за того, что структура команд чаще всего совсем не похожа на структуру тех действий, которыми пользователь предполагает описывать алгоритмы решения своих задач. Если составлять программы в машинных командах, пользователю неизбежно придется моделировать какими-то последовательностями команд каждое из своих действий. Но у этого способа программирования есть одно несомненное достоинство: он позволяет создавать *самые эффективные программы*. Объясняется это тем, что в данном случае имеется возможность тщательным образом учитывать совместно конкретные особенности как компьютера, так и задачи.

Рядовому пользователю не приходится программировать в машинных командах. У тех же специалистов, которым требуется создавать эффективные программы, такая потребность может появиться. Например, она часто возникает у разработчиков библиотек стандартных программ широкого назначения. Чтобы в какой-то мере облегчить труд таких специалистов, им предлагается программировать в некоторой системе символьного кодирования, называемой *автокодом*. По существу, это система тех же машинных команд, но с более удобной символикой. Кроме этого, в автокод могут быть введены дополнительные инструкции, моделирующие простейшие и наиболее часто встречающиеся комбинации машинных команд. Конечно, на компьютере имеется компилятор, который переводит программы, написанные на автокоде, в машинный код.

Автокод — это простейший язык программирования или, как его называют иначе, язык самого *низкого* уровня. Широко программировали на нем только на первых компьютерах. Массовое программирование на автокоде прекратилось по следующим причинам. Как отмечалось, программирование на нем очень трудоемко. Поэтому над автокодом стали создаваться различные языковые надстройки, называемые языками *высокого* уровня. Основная задача этих языков заключается в облегчении процесса программирования путем замены непонятных инструкций автокода другими инструкциями, более приближенными к пользовательским. Вершиной языков высокого уровня являются *проблемно-ориентированные* языки. Их инструкции позволяют пользователям писать программы просто в профессиональных терминах конкретных предметных областей. На каждом компьютере автокод, как правило, один. Языков высокого уровня, особенно проблемно-ориентированных, достаточно много. Компиляторы с них стали сложными иерархическими программными системами. Обратной стороной упрощения языков

программирования стало усложнение алгоритмов преобразования программ в машинный код и, как следствие, *потеря программистом возможности контролировать эффективность создаваемых им программ.* Это исключительно важное обстоятельство, и мы будем возвращаться к нему неоднократно, принимая во внимание нашу нацеленность на решение больших задач.

С точки зрения пользователя программировать на автокоде не только трудно, но и нецелесообразно из-за его ориентации на конкретный компьютер или, как говорят по другому, *компьютерной зависимости.* На разных моделях компьютеров автокоды разные. Универсального транслятора, переводящего программу из одного автокода в другой, не существует. Поэтому, переходя от одного типа компьютеров к другому, пользователю приходится переписывать заново все программы, написанные на автокоде. Этот процесс не только сверхсложный, но и крайне дорогой. Следовательно, одной из важнейших задач, стоящих перед разработчиками языков программирования высокого уровня, было создание языков, не зависящих от особенностей конкретных компьютеров. Такие языки появились в большом количестве. К ним относятся, например, Algol, Fortran, C и др. Они так и называются — *машинно-независимые*, что вроде бы подчеркивает их независимость от компьютеров. Еще их называют *универсальными*, что указывает на возможность описывать очень широкий круг алгоритмов.

Создание машинно-независимых языков программирования привело к появлению на рубеже 50—60-х годов прошлого столетия *замечательной по своей красоте идеи.* Традиционные описания алгоритмов в книгах нельзя без предварительного исследования перекладывать на любой компьютерный язык. Как правило, они неточны, содержат много недомолвок, допускают неоднозначность толкования и т. п. В частности, из-за предполагаемых свойств ассоциативности, коммутативности и дистрибутивности операций над числами в формулах отсутствуют многие скобки, определяющие порядок выполнения операций. Поэтому в действительности книжные описания содержат целое множество алгоритмов, на котором разброс отдельных свойств может быть очень большим. В результате, как правило, трудоемких исследований из данного множества выбирается какой-то *один алгоритм*, обладающий вполне приемлемыми характеристиками, и именно он программируется.

В программе любой алгоритм всегда описывается *абсолютно точно.* В его разработку вкладывается большой исследовательский труд. Чтобы избавить других специалистов от необходимости его повторять, надо этот труд сохранять. Программы на машинно-независимых языках высокого уровня как раз удобны для выполнения функций хранения. Идея состояла в том, чтобы на таких языках накапливать багаж хорошо отработанных алгоритмов и программ, а на каждом компьютере иметь компиляторы, переводящие программы с этих языков в машинный код. При этом вроде бы удавалось ре-

шить сразу две важные проблемы. Во-первых, сокращалось дублирование в программировании. И, во-вторых, автоматически решался вопрос о *портабельности* программ, т. е. об их переносе с компьютера одного строения на компьютер другого строения. Вопрос, который был и остается весьма актуальным, т. к. компьютеры существенно меняют свое строение довольно часто. При реализации данной идеи функции по адаптации программ к особенностям конкретных компьютеров брали на себя компиляторы. Как следствие, пользователь освобождался от необходимости знать устройство и принципы функционирования как компьютеров, так и компиляторов.

В первые годы идея развивалась и реализовывалась очень бурно. Издавались многочисленные коллекции хорошо отработанных алгоритмов и программ из самых разных прикладных областей. Был налажен широкий обмен ими, в том числе, на международном уровне. Программы действительно легко переносились с одних компьютеров на другие. Постепенно и пользователи отходили от детального изучения компьютеров и компиляторов, ограничиваясь разработкой программ на машинно-независимых языках высокого уровня. Однако радужные надежды на длительный эффект от реализации обсуждаемой идеи довольно скоро стали меркнуть.

Причина оказалась простой и оказалась связана с несовершенством работы большинства компиляторов. Конечно, речь не идет о том, что они создают неправильные коды, хотя такое тоже случается. Дело в другом — *ни один из компиляторов не гарантирует эффективность получаемых кодов.* Вопрос об эффективности редко возникает при решении небольших задач. Такие задачи реализуются настолько быстро, что пользователь не замечает многие проблемы. Но различные аспекты эффективности программ, в первую очередь, скорость их реализации и точность получаемых решений становятся очень актуальными, если на компьютер ставятся большие и, тем более, предельно большие задачи.

Целью производства новых компьютеров является предоставление пользователям возможности решать задачи более эффективно, в первую очередь, более быстро. Поэтому, разрабатывая программы на языках высокого уровня, пользователь вправе рассчитывать на то, что время их реализации будет уменьшаться примерно так же, как растет производительность компьютеров. Кроме этого, он, конечно, предполагает, что программы на машинно-независимых языках будут действительно не зависеть от особенностей техники. Однако весь опыт развития компьютеров и связанных с ними компиляторов показывает, что в полной мере на это надеяться нельзя. Скорость реализации программ как функция производительности компьютеров оказалась на практике не линейной, а значительно более слабой, причем, чем сложнее устроен компьютер, тем она слабее. Программы на машинно-независимых языках показали зависимость от различных особенностей компьютеров. Постоянное же появление все новых и новых языков программирования *без обеспечения каких-либо гарантий качества компилируемых* программ лишь увеличивает сложность

решения многих проблем. На практике пользователь не может серьезно влиять на разработку языковой и системной среды компьютера. Но именно он ощущает на себе все ее огрехи. Ему и приходится с ней разбираться, хотя это совсем не его цель. Его цель — решать задачу.

Возникла странная ситуация. Пользователь самым прямым образом заинтересован в эффективности решения своих задач на компьютере. Для повышения эффективности он прилагает огромные усилия: меняет математические модели, разрабатывает новые численные методы, многократно переписывает программы и т. п. Но пользователь видит компьютер только через программное окружение, в первую очередь, через компилятор и операционную систему. Создавая и исполняя машинные коды, эти компоненты сильно влияют на эффективность реализации пользовательских программ. Однако с их помощью либо трудно, либо просто невозможно понять, *где находятся узкие места созданного машинного кода и что необходимо изменить в конкретной программе на языке высокого уровня, чтобы повысить ее эффективность.* Действительная информация, которую можно получить от компилятора и операционной системы после реализации программы, чаще всего оказывается настолько скудной и трудно воспринимаемой, что ею редко удается воспользоваться при выборе путей модернизации как самой программы, так и реализуемых ею алгоритмов.

Для пользователя вопрос взаимодействия с компьютером с целью поиска путей создания наиболее эффективных программ является очень актуальным. Но его решение почти полностью переложено на плечи самого пользователя. Штатное программное окружение компьютера не содержит никаких отладчиков, систем анализа структуры программ и т. п. Из описаний компилятора практически невозможно найти даже такую необходимую информацию, как *эффективность работы в машинном коде отдельных, наиболее значимых конструкций языка высокого уровня.* Есть много оснований упрекнуть создателей языков, компиляторов и операционных систем в недостаточном внимании к интересам разработчиков алгоритмов и программ. В таком упреке есть доля истины, т. к. интересы пользователей действительно затерялись среди многих других интересов системных программистов и перестали быть доминирующими. Однако надо также признать, что проблемы пользователей оказались совсем непростыми.

Одной из важнейших характеристик вычислительных процессов является точность получаемых результатов. Уже давно известно, что кроме общего описания процессов, на нее влияют форма представления чисел и способ выполнения операции округления. Эти факторы машинно-зависимые. Информация о них не воспринимается языками высокого уровня. Поэтому, строго говоря, такие языки даже по одной этой причине нельзя считать машинно-независимыми, т. к. на разных компьютерах могут допускаться разные представления чисел и разные способы выполнения округлений. В конечном счете, это может приводить, и на практике приводит, к достаточно

большому разбросу результатов реализаций одной и той же программы. Но тогда возникают многочисленные вопросы: "Какой смысл надо вкладывать в понятие машинно-независимый язык? Как разрабатывать алгоритмы и писать программы, чтобы хотя бы с какими-то оговорками их можно было рассматривать как машинно-независимые? Как трактовать результаты реализации программы с точки зрения точности?" И т. п.

Были предприняты значительные усилия, чтобы получить ответы на все эти вопросы. Унифицированы формы представления чисел. Разработаны стандарты на выполнение операции округления. Для многих алгоритмов получены оценки влияния ошибок округления. В целом же успехи невелики. Возможно, что главным достижением предпринятых усилий является осознание того, что при разработке численного программного обеспечения вопросы влияния ошибок округления нельзя игнорировать. Какое-то время казалось, что проблема точности является единственной принципиальной трудностью, связанной с реализацией алгоритмов на компьютерах разных типов. И хотя на легкое решение этой проблемы нельзя было рассчитывать, тем не менее, обозначились определенные перспективы. Постепенно как будто стала формироваться уверенность, что все же удастся создать на алгоритмических языках эффективное машинно-независимое численное программное обеспечение и, следовательно, довести до практического решения проблему портабельности программ.

Действительность пошатнула и эту уверенность. Было выяснено, что на множестве программ, эквивалентных с точки зрения пользователя по точности, числу арифметических операций и другим интересным для него характеристикам, разброс реальных времен реализации различных вариантов на конкретном компьютере остается очень большим. Причем не только для сложных программ, но и для самых простых. Например, программы решения систем линейных алгебраических уравнений с невырожденной треугольной матрицей, написанные на языке высокого уровня, могут давать существенно различные времена реализации просто в зависимости от того, осуществляются ли в них внутренние суммирования по возрастанию индексов или по их убыванию. Это означает, что пользователь *принципиально не может* выбрать заранее тот вариант программы на языке высокого уровня, который будет одинаково эффективен на *разных* компьютерах. Принципиально не может хотя бы потому, что такой вариант, скорее всего, *не существует*. Но это означает также, что языки программирования высокого уровня можно и по этой причине считать машинно-независимыми лишь условно.

Итак, чтобы получить оптимальный машинный код, необходимо, как минимум, предоставить компилятору описание всего множества вариантов программ, эквивалентных с точки зрения пользователя. Но на традиционных алгоритмических языках высокого уровня можно описать только один вариант. Отсюда вытекает два важных вывода. Во-первых, эффективно решить проблему портабельности программ, ориентируясь на использование

языков высокого уровня *без предоставления дополнительной информации*, невозможно. И, во-вторых, для построения программы, обеспечивающей оптимальный машинный код на конкретном компьютере, исключительно важно иметь возможность получения *квалифицированной и достаточно полной информации* со стороны компилятора и операционной системы на каждый прогон программы.

До тех пор пока такая возможность не будет реализована полностью, пользователям придется многократно переписывать свои программы в поисках оптимального варианта. Для уменьшения числа переписываний применяются различные приемы. Например, в программах выделяются *вычислительные ядра*, т. е. участки, на которые приходится основной объем вычислений. Эти ядра пишутся на автокоде. При переходе к новому компьютеру переписываются только они. Тем самым обеспечивается высокая эффективность программ. По такому принципу организованы такие популярные пакеты для решения задач линейной алгебры, как EISPACK, LINPACK, SCALAPACK и др. Вычислительные ядра в них организованы в специальную библиотеку программ BLAS, которая написана на автокоде и переписывается каждый раз при переходе к новому компьютеру. Как показал опыт, этот прием довольно дорогой и не всегда применим, особенно при создании больших программных комплексов. К тому же, как выяснилось, при переходе к компьютерам с принципиально иной архитектурой приходится переписывать не только вычислительные ядра, но и их оболочку.

Даже краткое обсуждение показало, что вопросы разработки *эффективно реализуемых и долго используемых программ* оказались исключительно трудными и интересными. Они потребовали не только создания языков высокого уровня, но и изучения структуры программ и алгоритмов, разработки новых технологий программирования, определения места и меры использования машинно-ориентированных языковых сред, совершенствования работы компиляторов и операционных систем и многого другого. Все эти вопросы никак не исчезают в связи с усовершенствованием компьютеров, а, наоборот, становятся все более и более актуальными.

В последние годы во всем мире усиливается внимание к использованию высокопроизводительной вычислительной техники. С одной стороны, растет число пользователей персональных миникластеров, с другой стороны, происходит концентрация мощных вычислительных ресурсов в центрах коллективного пользования и развитие инфраструктуры удаленного доступа с использованием средств телекоммуникаций.

С другой стороны, в настоящее время всеми осознается факт, что дальнейшее повышение производительности компьютеров только за счет улучшения характеристик элементов электроники достигло предела, определяемого физическими законами. Дальнейшее повышение производительности возможно лишь за счет распараллеливания процессов обработки информации. Однако оказалось, что построение параллельных вычислительных процессов, обеспечивающих достижение максимальной производительности, является важной самостоятельной проблемой.

Сокращение времени выполнения большого объема работ путем разбиения на отдельные работы, которые могут выполняться независимо и одновременно, используется во многих отраслях. В области вычислительной техники это достигается путем увеличения числа одновременно работающих процессоров. Вместе с тем известно, что перенос обычной программы на многопроцессорную вычислительную систему может не дать ожидаемого выигрыша в производительности. Более того, в результате такого переноса программа может работать медленнее.

Опыт показывает, что написать эффективную параллельную программу или приспособить уже имеющуюся последовательную программу для параллельных вычислений чрезвычайно трудно. Связано это с тем, что переход к использованию многопроцессорных систем характеризуется принципиально новым содержанием. В данном случае важнейшим оказывается структурный анализ алгоритма, выявление его внутреннего параллелизма. Этот анализ требует глубокого понимания существа задачи. Часто же проблема заключается в том, что программист не вполне ясно понимает алгоритм решения задачи, а математик не может поправить программу, т.к. не обладает достаточной квалификацией в области программирования.

Эта трудность может быть преодолена, если использовать модель представления алгоритма, которая понятна как математику, так и программисту. При этом математик и программист могут работать в значительной степени независимо, взаимодействуя лишь в рамках используемой модели.

Представляется, что такое «разделение труда» позволит существенно сократить сроки решения сложных задач. Это освобождает прикладного математика высокой квалификации от необходимости написания кодов для решения своих задач. Математик может выполнить структурный анализ алгоритма, выявить имеющийся в нем параллелизм и представить в виде модели, понятной программисту. С другой стороны, программист, не вникая в существо задачи, используя лишь модель параллельных вычислений, может написать весьма эффективный код.

Именно поэтому центральная идея изучения параллельного программирования состоит в том, чтобы изучать вопросы построения параллельных алгоритмов и вопросы программирования независимо.

Стоит перейти непосредственно к основным определениям, понятиям, целям и задачам параллельных вычислений.

Под параллельными вычислениями (parallel or concurrent computations) можно понимать процессы решения задач, в которых в один и тот же момент времени могут выполняться одновременно несколько вычислительных операций. Параллельные вычисления составляют основу суперкомпьютерных технологий и высокопроизводительных расчетов. Одновременные выполняемые операции должны быть направлены на решение общей задачи, и параллельные вычисления следует отличать от многозадачных (многопрограммных) режимов работы последовательных ЭВМ – вспомнить по курсу НУП, как устроен защищённый режим. Параллельные вычисления – это форма выполнения операций, при которой несколько вычислений выполняются одновременно – в течение перекрывающихся периодов времени; вместо последовательного – с завершением одного до начала следующего. Параллельные вычисления есть свойство системы — будь то программа, компьютер или сеть, — где для каждого процесса существует отдельная точка выполнения или «поток управления». Параллельная система — это система, в которой вычисления могут выполняться, не дожидаясь завершения всех остальных вычислений. Решение общей

задачи разбивается на решение подзадач, каждая из которых выполняется в один и тот же промежуток времени.

Необходимость параллельных вычислений обусловлена тем, что требуется попытаться опередить потребности быстродействия вычислений существующих компьютерных систем в различных областях: моделирование и анализ изменения климата, состояния атмосферы и изменения погоды, геномика и генная инженерия, проектирование интегральных схем, анализы загрязнения окружающей среды, создание новых лекарственных препаратов и методов лечения заболеваний, поиск по широкомасштабным базам данных, инженерное виртуальное проектирование и оптимизация и многое, многое другое. Вместе с этим рост производительности последовательных компьютеров всегда будет оставаться ограниченным по сравнению с параллельным, многопроцессорные вычислительные системы становятся всё доступнее, а самые высокопроизводительные процессоры являются многоядерными.

Значимость же параллельных вычислений обусловливается следующими факторами: принятие обоснованных решений практически в любой сфере человеческой деятельности с необходимостью предполагает проведение расширенного математического моделирования с тщательным исследованием возможных вариантов деятельности с помощью вычислительных экспериментов. При этом, появление столь радикально возросших возможностей суперкомпьютерных технологий позволяет разрабатывать углубленные математические модели, максимально точно описывающих объекты реального мира, и требующие для своего анализа проведения масштабных вычислений. Кроме того, существуют области приложений, в которых параллельные вычисления имеют особую значимость:

- Невозможность (недопустимость) натурных экспериментов: изучение процессов при ядерном взрыве или серьезных воздействий на природу
- Изучение влияния экстремальных условий (температур, магнитных полей, радиации и др.) — старение материалов, безопасность конструкций, боевое применение
- Моделирование наноустройств и наноматериалов
- Науки о жизни — изучение генома человека, разработка новых лекарственных препаратов и т.п.
- Науки о Земле — обработка геоинформации: полезные ископаемые; селевая, сейсмическая и т.п. безопасность, прогнозы погоды, модели изменения климата...
- Моделирование при разработке новых технических устройств — инженерные расчеты

Наконец, суперкомпьютерные технологии становятся одним из решающих факторов научно-технического прогресса и могут служить точно таким стимулом развития страны, как ранее были авиация, атом, ракетная техника и космос.

Перейдём к задачам и целям параллельного программирования. Вследствие повсеместного использования вычислительной техники бурно развивается направление численного моделирования. Численное моделирование является промежуточным элементом между аналитическими методами изучения и физическими экспериментами. Рост количества задач, для решения которых необходимо использовать параллельные вычисления, обусловлен:

возможностью изучать явления, которые являются либо слишком сложными для исследования аналитическими методами, либо слишком дорогостоящими или опасными для экспериментального изучения;

быстрым ростом сложности объектов моделирования (усложнение и увеличение систем);
возникновением необходимости решения задач, для которых необходимо проведение анализа сложного поведения (например, условий перехода, к так называемому, детерминированному хаосу);
необходимостью управления сложными промышленными и технологическими процессами в режиме реального времени и в условиях неопределенности;
ростом числа задач, для решения которых необходимо обрабатывать гигантские объемы информации (например, 3D моделирование).
При этом, использование численных моделей и кластерных систем, позволяет значительно уменьшить стоимость процесса научного и технологического поиска. Кластерные системы в последние годы широко используются во всем мире как дешевая альтернатива суперкомпьютерам. Система требуемой производительности собирается из готовых, серийно выпускаемых компьютеров, объединенных, опять же, с помощью серийно выпускаемого коммуникационного оборудования. Это, с одной стороны, увеличивает доступность суперкомпьютерных технологий, а с другой, повышает актуальность их освоения, поскольку для всех типов многопроцессорных систем требуется использование специальных технологий программирования для того, чтобы программы могли в полной мере использовать ресурсы высокопроизводительной вычислительной системы.
Создать программу для выполнения которой будут задействованы все ресурсы суперкомпьютера не всегда возможно. В самом деле, при разработке параллельной программы для распределенной системы мало разбить программу на параллельные потоки. Для эффективного использования ресурсов необходимо обеспечить равномерную загрузку каждого из узлов кластера, что в свою очередь означает, что все потоки программы должны выполнить примерно одинаковый объем вычислений.
Рассмотрим частный случай, когда при решении некоторой параметрической задачи для разных значений параметров время поиска решения может значительно различаться. Тогда мы получим значительный перекос загрузки узлов кластера. В действительности практически любая вычислительная задача выполняется в кластере неравномерно.
Несмотря на это, использование кластерных систем всегда более эффективно для обслуживания вычислительных потребностей большого количества пользователей, чем использование эквивалентного количества однопроцессорных рабочих станций, так как в этом случае с помощью системы управления заданиями легче обеспечить равномерную и более эффективную загрузку вычислительных ресурсов.
Необходимость получения высокой эффективности выполнения программ усложняет использование параллельных систем.
Ещё одна из важных целей использования параллельных вычислений состоит в необходимости моделировать процессы, которые в реальной жизни происходят одновременно – например, несколько клиентов, которые обращаются к серверу в один и тот же момент, или же множественная отправка сообщений в одну сеть с разными интервалами времени – примером тут служат промышленные CAN-сети, широко используемые для передачи данных между микроконтроллерами и устройствами сложных механизмов, например, автомобилей.
К преимуществам одновременных вычислений относятся:

Повышенная производительность программы - параллельное выполнение параллельной программы позволяет увеличить количество задач, выполняемых за заданное время, пропорционально количеству процессоров.
Высокая скорость отклика на ввод/вывод — программы с интенсивным вводом/выводом в основном ждут завершения операций ввода или вывода. Параллельное программирование позволяет использовать время, которое было бы потрачено на ожидание, для другой задачи.
Более подходящая структура программы — некоторые проблемы и проблемные области хорошо подходят для представления в виде параллельных задач или процессов.
Естественно, параллельное программирование не лишено недостатков. Одной из самых главных проблем является контроль конфликтов при использовании общих ресурсов, поскольку, как уже было сказано, при параллельных вычислениях все вычислительные ядра пользуются одной и той же памятью и, как возможное следствие – одними и теми же переменными. Поэтому в этой области необходимо обеспечение правильной последовательности взаимодействий или связи между различными вычислительными операциями и координация доступа к совместно используемым ресурсам. Потенциальные проблемы в данном случае включают состояние гонки, взаимоблокировки и нехватку ресурсов. Например, рассмотрим следующий алгоритм для снятия средств с расчетного счета, представленного общим балансом ресурсов:

```
bool withdraw(int withdrawal)
{
    if (balance >= withdrawal)
    {
        balance -= withdrawal;
        return true;
    }
    return false;
}
```

Предположим, что balance = 500, и два параллельных потока выполняют вызовы withdraw(300) и withdraw(350). Если строка проверки условия в обеих операциях выполняется до строки, в которой баланс непосредственно уменьшается, обе операции обнаружат, что balance >= withdrawal оценивается как true, и выполнение продолжится с вычитанием суммы снятия. Однако, поскольку оба процесса выполняют снятие средств, общая снятая сумма в конечном итоге будет больше, чем первоначальный баланс. Именно такого рода проблемы решаются с помощью управления параллелизмом и вышеупомянутой блокировке доступа к общим ресурсам.