Содержание

От издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com на странице с описанием соответствующей книги.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии данной книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Apress очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу dmkpress@gmail.com со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Об авторах

Майкл Восс – главный инженер группы архитектуры, графики и программного обеспечения в компании Intel. Он входил в команду разработки TBB еще до выхода версии 1.0 в 2006 году и был первым архитектором API потокового графа TBB. Также является одним из ведущих разработчиков программы Flow Graph Analyzer – графического инструмента для анализа потоков данных на однородных и гетерогенных платформах. Автор и соавтор свыше 40 печатных работ по вопросам параллельного программирования, часто консультирует заказчиков по широкому кругу проблем и помогает им эффективно использовать потоковые библиотеки Intel. До поступления на работу в Intel в 2006 году был доцентом факультета электронной и вычислительной техники имени Эдварда С. Роджерса в Торонтском университете. Получил степень доктора философии в школе электронной и вычислительной техники при университете Пердью в 2001 году.

Рафаэль Асенхо – профессор компьютерной архитектуры в Малагском университете, Испания. Получил степень доктора философии по технике связи в 1997 году и работал доцентом на факультете компьютерной архитектуры с 2001 по 2017 год. В 1996 и 1997 годах был приглашенным преподавателем в Иллинойском университете в Урбана-Шампейне, а в 1998 году – приглашенным научным сотрудником в том же университете. Также работал приглашенным научным сотрудником в Исследовательском центре Томаса Уотсона компании IBM в 2008 году и в компании Cray Inc. в 2011 году. Использует TBB начиная с 2008 года, а в последние пять лет занимается промышленным применением гетерогенных кристаллов, в которых TBB служит в качестве координирующего каркаса. В 2013 и 2014 годах приезжал в Иллинойский университет в Урбана-Шампейне для работы над кристаллами, объединяющими CPU и GPU. В 2015 и 2016 годах начал исследовательскую работу по кристаллам, объединяющим CPU и ППВМ (программируемая пользователем вентильная матрица, англ. FPGA), во время работы в Бристольском университете. Исполнял обязанности председателя на конференции ACM PPoPP'16 (по принципам и практике параллельного программирования) и был членом оргкомитета, а также членом программного комитета на нескольких конференциях по высокопроизводительным вычислениям (PPoPP, SC, PACT, IPDPS, HPCA, EuroPar и SBAC-PAD). В сферу его профессиональных интересов входят модели и архитектуры гетерогенного программирования, распараллеливание нерегулярного кода и оптимизация энергопотребления.

Джеймс Рейндерс – консультант, имеющий за плечами более тридцати лет опыта в области параллельных вычислений. Автор, соавтор или редактор девяти технических книг по параллельному программированию. Принимал участие в разработке ключевых элементов двух самых быстрых в мире компьютеров (номер 1 в списке Top500), а также многих других суперкомпьютеров и средств разработки ПО. В середине 2016 года Джеймс отметил 10 001 день (свыше 27 лет) работы в Intel, но и теперь продолжает писать, преподавать, программировать и консультировать в различных областях, связанных с параллельными вычислениями (высокопроизводительные вычисления и искусственный интеллект).

Благодарности

Два человека поддерживали этот проект с самого начала и до конца – Санджив Шах (Sanjiv Shah) и Херб Хинсторф (Herb Hinstorff). Мы благодарны им за поддержку, ободрение, а временами и ненавязчивое подталкивание.

По-настоящему героические усилия предприняли рецензенты, которые присылали нам содержательные и подробные отзывы на черновики отдельных глав. Благодаря высокому качеству их работы нам пришлось потратить больше времени на редактирование текста, чем первоначально планировалось. Но в результате книга стала лучше.

Круг рецензентов составляла элита пользователей и основных разработчиков TBB. Редко бывает, чтобы в доведении книги до ума принимало участие такое энергичное и благосклонно настроенное сообщество. Читатели книги должны знать этих людей поименно: Эдуapl Айгуаде (Eduard Ayguade), Кристина Бельдика (Cristina Beldica), Константин Бояринов, Хосе Карлос Кабалейро Домингес (José Carlos Cabaleiro Domínguez), Брэд Чемберлен (Brad Chamberlain), Джеймс Джен-Чань Чен (James Jen-Chang Chen), Джим Коуни (Jim Cownie), Сергей Диденко, Алехандро (Алекс) Дюран (Alejandro [Alex] Duran), Михаил Дворский, Рудольф Rudolf (Руди) Эйгенман (Rudolf Eigenmann), Джордж Элькоура (George Elkoura), Андрей Федоров, Алексей Федотов, Томас Фернандес Пена (Tomás Fernández Pena), Элвис Фефей (Elvis Fefey), Евгений Фиксман, Базилио Фрагуэла (Basilio Fraguela), Генри Гэбб (Henry Gabb), Хосе Даниэль Гарсиа Санчес (José Daniel García Sánchez), Мария Хесус Гарзаран (Maria Jesus Garzaran), Александр Гервеши (Alexander Gerveshi), Дарио Суарес Грасиа (Darío Suárez Gracia), Кристина Керманшахче (Kristina Kermanshahche), Янив Клейн (Yaniv Klein), Марк Лубин (Mark Lubin), Антон Малахов, Марк Маклафлин (Mark McLaughlin), Сюзан Мередит (Susan Meredith), Есер Мезиани (Yeser Meziani), Давид Падуа (David Padua), Никита Пономарев, Ануп Мадхусоодханан Прабха (Anoop Madhusoodhanan Prabha), Пабло Ребле (Pablo Reble), Арч Робисон (Arch Robison), Тимми Смит (Timmie Smith), Рубен Гран Техеро (Rubén Gran Tejero), Вазант Товинкере (Vasanth Tovinkere), Сергей Виноградов, Кайл Уилер (Kyle Wheeler) и Флориан Зитцельбергер (Florian Zitzelsberger).

Мы искренне благодарны всем помощникам и приносим извинения тем, кого забыли упомянуть.

Майк (а вместе с ним Рафа и Джеймс!) благодарят всех принимавших участие в работе над TBB на протяжении многих лет: многочисленных разработчиков в Intel, оставивших свой след в библиотеке; Алексея Куканова, который рассказывал нам о ее внутреннем устройстве; сообщество разработчиков программ с открытым исходным кодом; технических писателей и специалистов по маркетингу, которые трудились над документацией и распространением информации о TBB; технических консультантов и прикладных программистов, которые помогают пользователям применять TBB к их задачам; менеджеров, не дававших нам сбиться с пути; а особенно пользователей TBB, которые присылали отзывы о библиотеке и ее функциональности, подсказывавшие нам,

в каком направлении двигаться. А больше всех Майк благодарит свою жену Натали и детей, Ника, Али и Люка, за поддержку и терпение на протяжении вечеров и выходных, проведенных в работе над книгой.

Рафа благодарит своих аспирантов и коллег за советы о том, как понятнее донести идеи TBB: Хосе Карлоса Ромеро (José Carlos Romero), Франсиско Корбе-ру (Francisco Corbera), Алехандро Виллегаса (Alejandro Villegas), Денизу Андреа Константинеску (Denisa Andreea Constantinescu), Анжелес Наварро (Angeles Navarro). Особая благодарность Хосе Даниэлю Гарсиа за увлекательные и ин-формативные беседы по поводу C++11, 14, 17 и 20, а также Алексею Федотову и Пабло Ребле (Pablo Reble) за помощь с примерами применения OpenCL_node, и прежде всего своей жене Анжелес Наварро за поддержку и выполнение не-которых его обязанностей во время работы над книгой.

Джеймс благодарит свою жену Сюзан Мередит – без ее терпеливой и не-ослабной поддержки книга была бы невозможна. Ко всему прочему детальная правка, когда за красными чернилами иногда не видно было оригинального текста, сделала ее одним из самых ценных наших рецензентов.

Будучи соавторами, мы не находим слов, чтобы воздать должное друг другу. Майк и Джеймс много лет знают друг друга по работе в Intel, и большая удача, что они сошлись в этом проекте. Трудно выразить, как Майк и Джеймс ценят Рафу! Как же повезло его студентам иметь такого энергичного и знающего профессора! Без Рафы читать эту книгу было бы далеко не так приятно. Благо-даря знаниям Рафы о TBB книга стала гораздо лучше, а английским он владеет так хорошо, что не раз исправлял ошибки своих англоязычных коллег (Майка и Джеймса). Мы все трое работали над книгой с удовольствием и определенно подстегивали друг друга. Это было прекрасное сотрудничество.

Мы благодарны Тодду Грину (Todd Green), который привел нас в издатель-ство Apress. Спасибо Натали Пао (Natalie Pao) из Apress и Джону Сомоса (John Somoza) из Intel, которые цементировали отношения между Intel и Apress в этом проекте. Мы высоко ценим тяжкий труд всего коллектива Apress над заключением контракта, редактированием и производством.

Спасибо всем!

Майк Восс, Рафаэль Асенхо и Джеймс Рейндерс

Предисловие

Мыслите параллельно

Мы ставили себе целью сделать эту книгу полезной как начинающим, так и искушенным в параллельном программировании. Мы также хотели, чтобы книга была доступна как тем, кто владеет только программированием на C, так и тем, кто уверенно пишет на C++.

Для охвата столь широкой аудитории без «оболванивания» книги мы и написали это предисловие, чтобы уравнять правила игры.

Что такое TBB

TBB – это библиотека для написания параллельных программ на C++, ставшая самым популярным решением и могущая похвастаться отличной поддержкой. Она широко используется – и не без причины. Созданная более десяти лет назад, TBB прошла испытание временем и учитывалась при включении поддержки параллельного программирования в стандарт C++. Хотя C++11 содержит много добавлений, связанных с параллельным программированием, а C++17 и C++2x продвинулись еще дальше в этом направлении, TBB предлагает куда больше, чем стандарт языка. Первая версия TBB была выпущена в 2006 году, поэтому библиотека по-прежнему поддерживает компиляторы, предшествующие выходу C++11. Но мы упростили себе задачу, приняв современный взгляд на TBB и предполагая, что все функции, описанные в C++11, наличествуют. В наши дни дают такой совет: «если у тебя нет компилятора C++11, поставь его». Если сравнивать с книгой о TBB, вышедшей в 2007 году, то, на наш взгляд, C++11, а особенно поддержка лямбда-выражений, расширяют функциональность TBB, а также упрощают ее понимание и использование.

Проще говоря, TBB – лучший способ написать параллельную программу на C++, и мы полагаем, что с TBB ваша продуктивность резко возрастет.

Структура книги и предисловия

В книге четыре основные части.

I. Предисловие. Базовые сведения, полезные для понимания остальной части книги. Содержит обоснование модели параллельного программирования, выбранной в TBB, введение в параллельное программирование, вопросы локальности, кеши, векторные вычисления (набор команд SIMD) и основные средства C++ (сверх имеющихся в языке C), которые поддерживаются или используются TBB.

II. Главы 1–8. Собственно книга о TBB. Включает введение в TBB в объеме, достаточном для эффективного параллельного программирования.
III. Главы 9–20. Включает специальные темы, углубляющие понимание TBB и параллельного программирования. Рассматриваются тонкие нюансы того и другого.
IV. Приложения A и B и глоссарий. Собрание полезных сведений о TBB, которые могут показаться вам интересными, в том числе история (приложение A) и полное справочное руководство (приложение B).

Мыслите параллельно

Для незнакомых с параллельным программированием мы написали это введение, в котором излагаются базовые сведения, делающие книгу более понятной, полезной и независимой. Мы предполагаем только владение языком C на базовом уровне и знакомим с ключевыми элементами C++, которые TBB поддерживает и на которые опирается. Мы рассматриваем параллельное программирование с практической точки зрения, подчеркивая те черты, которые делают параллельные программы более эффективными. Надеемся, что для опытных программистов это предисловие станет полезным напоминанием о теорминологии и способах рассуждений, позволяющих извлекать максимум из параллельного оборудования.

Прочитав предисловие, вы сможете объяснить, что значит «мыслить параллельно» в терминах декомпозиции, масштабирования, корректности, абстрагирования и паттернов. Вы будете понимать, что ключом ко всему параллельному программированию является локальность. Вам раскроется философия программирования на уровне задач, а не на уровне потоков – *революционное достижение концепции параллельного программирования, поддерживаемой TBB*. Вы познакомитесь с теми элементами программирования на C++ сверх известного по программированию на C, которые необходимы для использования TBB.

Предисловие состоит из пяти частей:

1) объяснение мотивов, стоящих за TBB;
2) введение в параллельное программирование;
3) введение в локальность и кеши – аспект оборудования, который, на наш взгляд, неотделим от достижения максимальной производительности средствами параллельного программирования;
4) введение в векторизацию (набор команд SIMD);
5) введение в языковые средства C++ (сверх унаследованных от C), которые используются или поддерживаются библиотекой TBB.

Мотивы, стоящие за библиотекой TBB

Библиотека TBB появилась в 2006 году. Ее создали специалисты по параллельному программированию из компании Intel, и за плечами многих из них были десятки лет работы с моделями параллельного программирования, в т. ч. OpenMP. Многие члены команды TBB потратили годы, чтобы OpenMP могла добиться

впечатляющих успехов, для чего разрабатывали и поддерживали различные реализации OpenMP. Приложение А посвящено истории TBB и ее ключевых концепций, в т. ч. прорывной идее планировщиков с заимствованием задач.

Появившись на заре создания многоядерных процессоров, TBB быстро превратилась в самую популярную у программистов на C++ модель параллельного программирования. На протяжении первого десятилетия после рождения TBB впитывала в себя разнообразные дополнения, сделавшие ее очевидным выбором для параллельного программирования равно у начинающих и опытных пользователей. Будучи проектом с открытым исходным кодом, TBB получала отклики и дополнения со всего мира.

TBB продвигает революционную идею: параллельное программирование должно дать программисту возможность без колебаний выявлять места, подходящие для распараллеливания, а реализация базовой модели программирования (TBB) должна отображать его желания на аппаратные средства во время выполнения.

В основе важности и ценности TBB лежит понимание трех вещей: (1) программирование с применением задач, а не потоков; (2) модели параллельного программирования необязательно должны быть запутанными; (3) как добиться масштабируемости, высокой производительности и переносимости при работе с переносимыми моделями параллельного программирования с низкими накладными расходами, примером которых является TBB. Далее мы рассмотрим все три этих крайне важных аспекта! Можно с уверенностью сказать, что до того, как они стали краеугольными камнями эффективного и структурированного программирования, их важность долгое время недооценивалась.

Программирование с применением задач, а не потоков

Программировать параллельно всегда следует в терминах *задач*, а не *потоков*. В конце этого предисловия мы процитируем авторитетный и глубокий анализ этого положения Эдвардом Ли. В 2006 году он заметил: «Чтобы конкурентное программирование стало обыденностью, необходимо отказаться от потоков как модели программирования».

Параллельное программирование в терминах *потоков* – это упражнение на тему отображения приложения на некоторое число параллельных потоков выполнения на той машине, где выполняется программа. Параллельное программирование в терминах *задач* – это упражнение на тему выявления возможных мест для распараллеливания, после чего исполняющая среда (например, среда TBB) отображает задачи на оборудование во время выполнения, не вынуждая программиста усложнять логику приложения.

Логический *поток* исполняется аппаратным потоком в течение кванта времени, а в будущих квантах времени может быть назначен другому аппаратному потоку. Модель параллельного программирования в терминах потоков терпит провал, потому что часто используется как взаимно однозначное соответствие между логическими потоками и аппаратными потоками (например, процессорными ядрами). Аппаратный поток – это физическое свойство, от машины к машине меняется их количество, равно как и некоторые тонкие характеристики различных реализаций потоков.

Напротив, *задачи* представляют *возможные места* распараллеливания. Разбиение на задачи можно использовать по мере необходимости с учетом количества доступных аппаратных потоков.

Имея в виду эти определения, можно сказать, что программа, написанная в терминах потоков, должна отображать каждый алгоритм на конкретную систему, состоящую из аппаратного и программного обеспечения. Это не только отвлекает внимание, но и порождает целый ряд проблем, из-за которых параллельное программирование оказывается более трудным, менее эффективным и гораздо менее переносимым.

В то же время программа, написанная в терминах задач, допускает наличие механизма времени выполнения, например исполняющей среды TBB, который отображает задачи на реально имеющееся оборудование. Это позволяет не думать о том, каким количеством аппаратных потоков в действительности располагает система. Но важнее то, что на практике это единственный метод, позволяющий эффективно использовать вложенный параллелизм. Это настолько важная возможность, что мы будем возвращаться к ней в нескольких главах.

Компонуемость: параллельное программирование необязательно должно быть запутанным

Библиотека TBB обеспечивает *компонуемость* в параллельном программировании, а это меняет все. Компонуемость означает, что мы можем совместно использовать различные средства TBB без ограничений. А самое главное – она допускает вложенность. В частности, ничто не запрещает поместить один цикл `parallel_for` внутрь другого. Из цикла `parallel_for` можно также вызвать подпрограмму, внутри которой имеется другой цикл `parallel_for`.

Поддержка компонуемого вложенного параллелизма в высшей степени желательна, поскольку открывает больше возможностей для распараллеливания, а это, в свою очередь, позволяет создавать более масштабируемые приложения. Например, система OpenMP не является компонуемой относительно вложенности, т. к. каждый уровень вложенности может приводить к значительным накладным расходам и потреблению ресурсов, что станет причиной истощения ресурсов и аварийного завершения программы. Серьезность этой проблемы становится очевидной при попытке использовать библиотечную подпрограмму, содержащую параллельный код. В TBB подобной проблемы нет, поскольку она поддерживает компонуемость. Отчасти она решается благодаря тому, что TBB позволяет программисту указать места распараллеливания (задачи), а сама во время выполнения решает, как отобразить их на аппаратные средства (потоки).

Это важнейшее преимущество кодирования в терминах задач (доступный, но необязательный параллелизм (см. раздел об «ослабленной последовательной семантике» в главе 2)), а не потоков (принудительный параллелизм). Если бы цикл `parallel_for` считался обязательным, то вложенность привела бы к взрывному росту количества потоков вместе с ворохом проблем неконтролируемого потребления ресурсов, которые легко могут вызвать (и часто вы-

зывают) крах программы. Если же `parallel_for` рассматривается как доступный, но необязательный параллелизм, то исполняющая среда вправе использовать эту информацию при выборе наиболее эффективного отображения на аппаратные средства компьютера.

Мы привыкли ожидать компонуемости от языков программирования, но в большинстве моделей параллельного программирования это свойство утрачено (к счастью, TBB является исключением!). Рассмотрим, к примеру, предложения `if` и `while`. В языках C и C++ они могут сочетаться и вкладываться как угодно. Но представим себе, что это не так – что мы живем в мире, где функция, вызванная из предложения `if`, не может содержать предложения `while`! Сама мысль о таком ограничении кажется нелепой. TBB привносит такого рода компонуемость в *параллельное программирование*, разрешая произвольно сочетать параллельные конструкции без опасения вызвать проблемы.

Масштабируемость, производительность и погоня за переносимой производительностью

Быть может, самым важным преимуществом программирования с использованием библиотеки TBB является то, что она помогает создавать приложения с переносимой производительностью. Мы определяем *переносимую производительность* как характеристику, благодаря которой у программы оказывается похожий «процент пиковой производительности» на различных машинах (с различным оборудованием, разными операционными системами или тем и другим сразу). Мы хотели бы, чтобы высокий процент пиковой производительности имел место на самых разных машинах без необходимости изменять код.

Мы также хотели бы видеть 16-кратный прирост производительности на машине с 64 ядрами по сравнению с четырехъядерной машиной. По различным причинам такое идеальное ускорение на практике почти никогда не наблюдается (но никогда не говори никогда: в некоторых ситуациях благодаря увеличению совокупного размера кеша наблюдается даже более чем идеальное ускорение – это называется *сверхлинейным ускорением*).

Что такое ускорение?

Изначально ускорение определяется как время последовательного выполнения программы, поделенное на время ее параллельного выполнения. Если обычно моя программа работает 3 с, а на четырехъядерном процессоре всего 1 с, то говорят, что ускорение трехкратное. Иногда употребляют термин *эффективность* – это ускорение, поделенное на количество процессорных ядер. Таким образом, трехкратное ускорение эквивалентно эффективности распараллеливания 75 %.

Идеал – 16-кратный прирост производительности при переходе от четырехъядерной машины к 64-ядерной – называется линейной, или идеальной, *масштабируемостью*.

Для его достижения необходимо обеспечить полную занятость всех ядер при увеличении их количества – цель, требующая большого уровня доступного па-

раллелизма. Понятие доступного параллелизма мы более внимательно рассмотрим ниже при обсуждении закона Амдала и следствий из него.

Пока же важно знать, что TBB поддерживает высокопроизводительное программирование и помогает в достижении переносимой производительности. Высокая производительность проистекает из того, что TBB не вносит почти никаких накладных расходов, что позволяет беспрепятственно масштабировать программу. Переносимая производительность позволяет приложению задействовать весь доступный параллелизм, предлагаемый современными компьютерами.

Уверенно делая такие заявления, мы предполагали, что незначительность дополнительных накладных расходов на планирование задач обеспечивает максимальную эффективность выявления и использования возможностей распараллеливания. У этого предположения есть один изъян: если мы напишем программу, идеально соответствующую оборудованию, но без возможности динамической подстройки, то, возможно, сумеем увеличить производительность на несколько процентов. Традиционная модель высокопроизводительных вычислений (High-Performance Computing – HPC), применявшаяся для программирования интенсивных массивно параллельных вычислений на самых больших в мире компьютерах, давно уже обладала такой характеристикой. Разработчик, привыкший к HPC, использующий систему OpenMP со статическим планированием и довольный ее производительностью, вероятно, обнаружит, что TBB со своей динамичностью несколько снижает производительность. Но преимущества такого статического планирования по различным причинам постепенно сходят на нет. По мере усложнения программ на основе модели HPC требуется поддержка вложенного и динамичного параллелизма. Мы видим это во всех аспектах HPC-программирования: появление больших мультифизических моделей, включение искусственного интеллекта (ИИ) и использование методов машинного обучения (МО). Одна из главных причин дополнительной сложности – применение разнообразного оборудования, в результате чего на одной машине появляется гетерогенная вычислительная среда. TBB предлагает средства справиться с этими сложностями, в т. ч. потоковый граф, который мы будем изучать в главе 3.

Очевидно, что для эффективного параллельного программирования необходимо разделять выявление мест распараллеливания в форме задач (функция программиста) и отображение задач на аппаратные потоки (функция реализации модели программирования).

Введение в параллельное программирование

Прежде чем сорвать покров тайны с терминологии и основных понятий параллельного программирования, сделаем смелое заявление: параллельное программирование интуитивно более понятно, чем последовательное. Параллелизм окружает нас в повседневной жизни, делать свои дела шаг за шагом – роскошь, которую мы далеко не всегда можем себе позволить. Параллелизм не является чем-то неизведанным и не должен быть таким в программировании.