Я буду откровенен, я прекрасно понимал о чем пишет Эдвард Ли, ровно до момента появления формул, описывающих работу последовательных программ. Однако, спустя всего пару страниц, пример с кодом на джаве стал плавно прояснять что хочет донести автор статьи.
Статья “The Problem With Threads” посвящена, как не странно, проблеме параллельного, а именно многопоточного программирования.

В первой половине (разделы 1-4) доклада, начиная с предисловия, Эдвард Ли рассказывает о том насколько многопоточное программирование ненадежно и как оно вредит разработчикам, приложениям которые они пишут и конечно конечным пользователям. Нам говорят о том, что: зачастую надежность и предсказуемость работы программ более важны, чем производительность и гибкость; “Нужно иметь возможность добавить недетерминизм при необходимости, а не возможность убрать его при “ненужности”.”; потоки являются крайне недетерминированными компонентами; а также подчеркивают, на примере реализации паттерна “наблюдатель” на языке Java, каким тяжелым или неочевидным может быть решение даже самых простых задач при многопоточном подходе. Автор всячески сгущает краски и пару раз называет программистов, пищущих многопоточный код, безумцами, давая четкое определение безумию - “Безумие - это повторений одних и тех же действий, раз за разом, в надежде на изменение результата”.
Во второй половине доклада Эдвард рассказывает как можно, по крайней мере пытаться, бороться с всемозможными ошибками вроде исключений или дедлоков. Он упоминает проект Ptolemy Project и ненадежность (если не невозможность) полного тестирования продукта, несмотря на высокое качество разработки; транзакции, как программные, так и аппаратные; координационные языки и расширение уже существующих языков, таких как C, C++, Java; promises и futures. Ну а в конце Эдвард снова упоминает теорию вычислений и приводит формулу моделирования параллельных вычислений.

Итак, проанализировав данный весьма увлекательный доклад, я могу выразить следующие мысли:
Тяжело не согласиться с автором в том, что параллелизм действительно опасен, так как часто добавление любой абстракции в программировании приводит к снижению досконального понимания тех или иных процессов. Добавляя различные классы, интерфейсы, разного уровня вложенности методы и другие структуры языка в ваш код, Вы тем самым повышаете уровень абстракции, то есть отдаляетесь от фактических функций, которые выполняет ваша программа или конкретные её части. И это всё в последовательном, однопоточном программировании. Как только речь заходит о параллелизме, шансы возникновения ошибок начинают расти с невероятной скоростью, так как отслеживать и без того довольно абстрактные функции, в нескольких потоках сразу, становится практически невозможно.
Стоит ли вообще это обсуждать если до сих пор находятся баги, НИКАК не связанные с многопоточностью, в системах, работающих уже много лет? Думаю тут всё довольно понятно - любая система должны быть максимально, насколько это возможно, надежной; параллелизм понижает надежность и прозрачность системы => вредит конечному продукту (в той или иной степени).
Ну и конечно, хочется добавить, что, на самом деле, далеко уйти от параллелизма у нас не получится, так как различные компании, производящие

многоядерные процессоры, например Intel, всячески продвигают идеи многопоточного программирования.

Мы живём в интересное время, когда технологии развиваются невероятно быстро и совсем непонятно чего ждать дальше, в связи с чем после всего сказанного выше хочется подвести итог следующим образом:
я, пожалуй, предпочту просто сидеть и наблюдать за тем, что же будет происходить с параллелизмом и многопоточным программированием. Будут ли отказывать целые системы и сервисы из-за модернизации и оптимизации железа, и будут ли новые поколения программистов злоупотреблять многопоточностью.