1. **Is the method for sum calculation CPU-consuming or IO-bound**

Вообще, **CPU-consuming**. Минутка размышлений:

Программа привязана К ЦП, если скорость выполнения зависит от ЦП, т.е .если бы ЦП был быстрее, то и скорость выше (т.е. большую часть времени программа тратит время на использование ЦП, выполнение вычисления)

Привязка к программе, как IO-bound в свою очередь если система ввода-вывода будет работать быстрее(диск, сеть и т.п.). Например, программа, которая просматривает огромный файл дабы найти некоторую, малую часть данных. Она будет привязана к вводу-выводу, поскольку узким местом будет являться чтение данных с диска (соотношение мб/с, ссд и т.п.)

Еще кстати есть Memory-bound и cache-bound. Ограничение памяти, т.е. скорость с которой идёт процесс с ограничем доступной памятью и скоростью этой памяти. Кэш привязка означает, что скорость выполнения будет ограничена объемом и скоростью кэша. Задача, которая обрабатывает больше данных, чем помещается в кэш – будет привязана к кэшу.

И по фату по идее выходит так: IO-Bound < Memory-Bound < Cache-Bound < CPU-Bound

К чему я это все – наткнулся на пример. Чел решил потестать и заюзал многопоточку для расчета суммы итераторов в больших циклах. И как он пишет – распараллеливание, разделеие массива поровну для каждого из ядер имеет ограниченную полезность на современных ПК. Как он утверждает – вычисления не стали в 8 раз быстрее на 8-ми ядерном процессоре. По какой-то причине оптимальным было 2/3 потока, а добавление больших просто замедляло процесс. И причиной было использование одной шины памяти соединяющуюся с RAM. Т.е. узким место становится шина памяти, а не процессор. Единственный способ ускорить вычисления: увеличить скорость доступа к памяти. **Например многоканальную память заюзать, как я понял))) и вот поэтому существует класс Parallel, который круто оптимизирет и подбираем количество потоков для распараллеливания))**

За сим, если я правлиьно понимаю: на малых объемах мы привязаны к CPU, но если вычисления будут громоздкими, то мы уже зависим от памяти, нежеле от CPU.

1. **Does your code use async void? Why?**

Нет. Как я понял от него нету смысла, потому что если мы юзаем void, вызывной метод не знает, когда задача будет окончена и благодаря этому мы можем хватануть deadlock, либо прилажение просто начнет работать некорреткно. Единственное возможное применение – использовать войд при собитийных сработках. Т.е. вызывать на событиях.

1. **Is there any alternative for Thread.Sleep() when it comes to artificial delay? What is the benefit of it?**

Начнем с того, что thread.sleep() не гуд, потому что он вытесняет ресурсы потока из кэша. И подргужает их заново после пробуждения. По сути погоды он не делает, но может стать проблемой производительности в хай-лоад ситуациях.

Можно юзать manualReset:

new System.Threading.ManualResetEvent(false).WaitOne(1000). Т.е. мы делаем обработчик событий, который никогда не будет засетаплен, он просто подождет установленное время WaitOne.

Для специфических сценариев можно также юзать таймеры:

Var timer = new system.threading.timer(

(X) => DoWork(), null, 1000, 3000)

);

Вместо thread.sleep можно юзать чудо таски (Task.Delay()). Он примичателен тем, что его можно юзать асинхронно. Он чаще всего так и юзается.

1. **How do you handle cancellation in your code? Do you use OperationCancelledException or TaskCanceledException? What is the difference between them?**

Я используюCancellationTokenSource и вызываю cts.cancel(), в самом методе отслеживаю состояние токена по проперте IsCancellationRequested.

А сама разница OperationCancelledException и TaskCanceledException заключается в token.ThrowIfCancellationRequested(). Этот метод проверяет отмену и если требуется – генерит исключение OperationCancelledException, а не TaskCanceledException.

Исключение TaskCanceledException сработает если мы кэнсельнём таску до того, как она начнет выполняться.

1. **Do you throw cancellation exception when the cancellation is triggered right before the work is done? What is the best way to handle such a situation?**

Нет, я просто возвращаю результат на момент отмены действия. На данный момент, как я понял – бенефит исопльзования исключений в том, что мы можем запихнуть таску в конструктор и выцепить больше инфы из таски в вызывающем методе. Это наверное и будет best way.

1. **What goes first – input validation or cancellation checks? Why?**

Input validation. Это поможет выловить ошибки в вызывающем коде.

1. **What can you tell about CancellationToken.CanBeCanceled? Could you describe a situation where it can be useful?**

Пропертя, которая скажет нам может ли таска быть отменена. Единственный бенефит от использования по моему мнению будет в том, что прежде, чем проперять значение свойства IsCancellationRequested – мы можем проверить «а может ли эта таска быть кэнсельнута» ))