**Краткий путеводитель по многопоточным режимам МТ**

По материалам http://avisynth.org/mediawiki/Internal\_functions/Mu...eading\_functions , http://avisynth.org/mediawiki/MT\_support\_page и на основе жизненного опыта. Предназначено для широкой аудитории, к формулировкам не придираемся.

Если используете Avisynth 2.6 MT сборки SEt (http://forum.doom9.org/showthread.php?t=148782), то в указанной сборке МТ встроен на уровне ядра Avisynth, поэтому не загружайте фильтр МТ отдельно с помощью LoadPlugin в своих скриптах (обратное целесообразно только в том случае, если возникает крайняя необходимость в вызове функции типа MT("фильтр",2,2), которая не встроена в ядро Avisynth МТ)!

На особенностях работы МТ с использованием настроек вида MT("фильтр",2,2) останавливаться не собираюсь -- будем разбираться с setmtmode, который не вызывает побочных эффектов при правильном его применении.

**setmtmode(\*, 4)** - Вторая цифра (после запятой) определяет количество потоков, которые будут использоваться для обработки ВСЕГО скрипта (в данном примере - 4 ядра без НТ или 4 логических потока с НТ). Такую комбинацию имеет смысл использовать только в начале скрипта, поскольку при дальнейшей его работе количество потоков невозможно поменять. По этой причине в середине скрипта обычно встречается setmtmode(\*) с указанием только первой цифры, которой определяется режим работы МТ, т.е. "mode", о чем идет речь дальше:  
**setmtmode(1)** - Самый быстрый режим, но работает только с некоторыми фильтрами, в частности: MPEG2Source, Trim, GreyScale, RemoveDirt.  
**setmtmode(2)** - Практически самый распространенный режим, работает с большинством фильтров. Работа каждого фильтра распределяется между доступными потоками ПРОЦЕССОРА (физическими и/или логическими), поэтому для обслуживания очереди задач используется больше ресурсов оперативной памяти (иногда требуется тонкая настройка SetMemoryMax).  
**setmtmode(3)** - Предназначен в первую очередь для фильтров, которые не работают нормально в mode 2, в том числе для загрузчиков медиа-источников. Позволяет использовать по одному потоку ПРОЦЕССОРА для каждого фильтра, поэтому работает медленней, чем setmtmode(2), но менее требователен к ресурсам памяти. Данный режим весьма полезен для жадных к ресурсам фильтров, способных самостоятельно (без помощи МТ) разделять свою работу на несколько ВИРТУАЛЬНЫХ потоков внутри потока процессора.  
**setmtmode(4)** - Это комбинация двух режимов - setmtmode(2) и setmtmode(3) - удовлетворяющая, соответственно, требования подавляющего большинства фильтров. Недостатки: является одновременно медленным и требовательным к ресурсам памяти; может запросто подвесить систему при использовании "жадных" фильтров.

Теоретические размышления (цифры карикатурные): На 4-ядернике с НТ используется "жадный" фильтр, который в зависимости от поставленной задачи разделяет свои процессы на 2-20 виртуальных потоков. При максимальной загрузке 20 потоками в режиме setmtmode(3), процессор может распределить такую работу по свободным ресурсам сам, без помощи фильтра МТ. В режиме setmtmode(2) виртуальных потоков внутри одного потока процессора может создаться больше благодаря фильтру МТ (минимум 40, а то и все 80-100), процессор будет вынужден искать и выделять ресурсы остальных своих 7 потоков, ему уже становится тяжеловато. При включении setmtmode(4) может быть создано одновременно 40-100 виртуальных потоков "жадного" фильтра в каждом из свободных потоков процессора (т.е. 40...100 умножаем на 2...6). И куда дальше процессор будет распределять эти потоки?... И какого размера должен быть кэш процессора, чтобы совладать с очередью, растущей в геометрической прогрессии?

**setmtmode(6)** - Это модифицированная версия режима setmtmode(5) со всеми его недостатками и достоинствами, но работающая чуть-чуть быстрей.  
**setmtmode(5)** - Самый медленный режим. Не устанавливает какие-либо ограничения, поэтому используется в основном перед критичными в этом вопросе загрузчиками медиа-источников. Предназначен для фильтров, не требующих линейного обслуживания фреймов (когда не требуется строгий порядок подачи фреймов: 0,1,2,...,последний).  
Также рекомендуется к использованию перед fft3dgpu, NLMeansCL и Deathray во избежание артефактов обработки видеокартой (инф. Tempter57).  
В связи с тем, что подавляющее большинство приложений предусматривает именно линейную подачу фреймов, **загрузка процессора при setmtmode(5) значительно меньше, чем работа без МТ вообще**, а в зависимости от поставленных задач обработка с setmtmode(5) может оказаться медленней, чем даже на стареньком 1-ядерном Athlon XP (безо всяких МТ разумеется).  
Следует отметить, что все обращения к скриптам Avisynth обрабатываются в обратной последовательности (снизу вверх). Таким образом, например, если составить такой скрипт:

setmtmode(2)  
setmemorymax(1024)  
source = last  
super = source.MSuper(...)  
.....  
setmtmode(5)  
Temporalsoften(...)  
setmtmode(2)  
ContraHD(last,source,...)

то ContraHD будет простаивать в ожидании Temporalsoften, а результаты работы MSuper и ContraHD мы получим не раньше, чем отработает свою задачу Temporalsoften в очень медленном режиме setmtmode(5).  
В связи с тем, что обычно пользователю не известны устройство и внутренние алгоритмы работы фильтров (в частности, как именно они используют ресурсы компьютера), при использовании режимов setmtmode весьма благоразумным является измерение скорости работы скриптов с помощью сторонних утилит, не зависящих от Avisynth, например, с помощью **AVSMeter** (<http://forum.doom9.org/showthread.php?t=162155>). Измеритель скорости запускается очень просто: перетаскиваем файл avs-скрипта мышкой на экзешник AVSMeter, ждем пару минут для набора статистики FPS (кадров в секунду), после чего проверка может быть прервана пользователем. При этом каждое изменение параметров setmtmode и setmemorymax проверяется на скорость выполнения, после чего варианты настроек сравниваются и выбирается наиболее оптимальный. Такой подход позволяет сэкономить уйму времени: потратив не более 20-30 минут на предварительные настройки и оптимизацию с помощью AVSMeter, можем избежать нескольких часов или суток лишней обработки в неоптимизированном режиме. ВАЖНО: для успешного использования в **64-бит**ной системе подходит версия AVSMeter не ниже **1.18** (<http://doom10.org/index.php?topic=1817.msg10487#msg10487>).

Подстановку значений SetMemoryMax и количества потоков (при их подборе и распределении между Avisynth и кодеком) можно автоматизировать с помощью весьма полезного vbs-скрипта, составленного **unreal666**: <http://rutracker.org/forum/viewtopic.php?p=53645942#53645942> (для продвинутых пользователей). Скрипт предназначен для взаимодействия с avs2pipemod, поэтому потребуется внести в него изменения самостоятельно, если используете другие подобные программы для кодирования.

**Возможные причины частичной загрузки процессора (<=60%) в режиме МТ, и способы ее устранения:**

1) Неверное применение режимов setmtmode(5) и setmtmode(6), а иногда и setmtmode(4) в тех местах скрипта (перед теми фильтрами), где больше подходят режимы 1-3 (изложено выше в данном сообщении);  
2) Неверный подбор параметра SetMemoryMax() при использовании режимов setmtmode(2) и setmtmode(4).  
Вообще параметр SetMemoryMax находится в подобной зависимости даже без МТ, но при указанных режимах МТ может иметь критическое значение. SetMemoryMax() -- это ограничитель, установка максимума оперативной памяти, которая будет выделена для работы фильтров Avisynth конкретного скрипта. Здесь надо выдержать БАЛАНС между потребностями Ависинта и потребностями кодека. Если мы установим SetMemoryMax слишком большим, то фильтры Avisynth могут вдруг забрать все свободные ресурсы памяти, после чего сами же будут простаивать в ожидании кодировщика, который будет тормозить весь процесс из-за нехватки памяти для кодирования. Если мы установим SetMemoryMax слишком низким, то возникнет обратная ситуация (что и происходит в большинстве таких случаев) -- кодировщик будет простаивать в ожидании фильтров, требующих больше памяти для завершения своих операций. При этом, в последней ситуации могут возникать критические СИСТЕМНЫЕ ОШИБКИ, поскольку в отличие от кодеков, адаптирующихся к объему доступных ресурсов, фильтры Avisynth способны создать очередь заданий, которую компьютерное "железо" не способно обработать физически;  
*Приведенные выше проблемные ситуации можно спокойно обойти, если применять AVSMeter или подобные утилиты при подготовке скриптов. Однако бывают ситуации, которые невозможно разрешить предложенным способом. Например:*  
3) Блокирование потоков процессора, доступных кодеку изначально. Важно понимать следующее применительно к многопоточным компьютерным системам (не знаю, ко всем ли кодекам относится, не вижу смысла проверять):

При запуске процесса кодирования, кодек получает установку на использование конкретного количества потоков (например 4 физических/логических потока процессора). Для выполнения своих задач кодек обычно имеет приоритет не выше "Normal", поскольку более высокие приоритеты могут приводить к зависанию системы, а зачастую его потоку назначается приоритет "Low" или "Very low" по умолчанию. Если в процессе кодирования стартуется системой или мы сами запускаем какое-либо приложение, предполагающее хотя бы временное получение более высоких приоритетов по сравнению с работающим кодеком, то в результате у кодека автоматически отбираются все доступные потоки, кроме одного необходимого, или то количество потоков, которые явно требует приложение с более высоким приоритетом. Затем, когда система обнаруживает, что вновь запущенному приложению достаточно для работы 1 потока к примеру, то она возвращает кодеку остальные 2 свободных потока, или освобождает для кодека все изначальные 3 потока по завершению работы упомянутого приоритетного приложения.  
Однако не следует забывать, что Avisynth в режиме МТ занимает ресурсы компьютера динамически по мере своих потребностей. При этом, потребности фильтров Avisynth проявляются в очереди задач компьютера конечно же раньше, чем финализирующие потребности кодека. Таким образом, после запуска приложения, отобравшего в силу своего приоритета несколько потоков у кодека, такие освободившиеся позже потоки могут быть не возвращены кодеку, потому что будут динамически заняты МТ-фильтром Ависинта.

Предотвратить эту проблему возможно следующими способами:

-- не запускайте какие-либо приложения, если кодируете с предварительным использованием Ависинта в МТ-режиме (если даже приложению явно не дается более высокий приоритет, то при его запуске могут стартоваться системные службы с такими приоритетами);  
-- если все-таки крайне необходимо делать что-то параллельно с кодированием, то запустите все необходимые для работы приложения до старта процесса кодирования, и до его окончания не завершайте их работу;  
-- по возможности деактивируйте процессорный режим HT (Hyper-Threading) на компьютере, где обычно производится кодирование;  
-- если по каким-либо причинам отключать HT не желательно, то хотя бы четко указывайте кодеку, что он может использовать количество потоков не больше количества физических ядер процессора (если ядер 4, то и потоков кодеку поставить 4, а не используемое по умолчанию например в Xvid4PSP5 количество потоков 6 [4\*1,5]);  
-- в случае продолжения проблемы или при невозможности использования одного из 4 вариантов выше, попробуйте радикально решить проблему БАЛАНСа, но уже применительно к распределению потоков между кодеком и Avisynth, т.е. изначально задайте фильтрам Ависинта и кодеку количество потоков, которое в сумме не превышает количество физических ядер процессора (например для 6-ядерника: setmtmode(\*, 3) в начале скрипта avs и 3 потока кодеку; или setmtmode(\*, 4) и 2 потока кодеку);

4) Отсутствие функции Distributor (без нее работа происходит в однопоточном режиме) в составе кодировщика или взаимодействующей с ним программы (случай в последнее время редкий, но всё же). Попробуйте прописать в самом конце своего avs-скрипта вызов функции Distributor(). Если загрузка процессора увеличится, то рекомендуется поступать так же в дальнейшем. Если же такое действие приведет к критической ошибке, то прописывать Distributor() в конце скрипта не следует.