**一、前期算法分析工作**

搞清肝脏分割算法的处理流程，调用哪些子模块，调用逻辑是怎样的。肝脏分割子模块在界面上需输入两个阈值，可选输入滚球半径和平滑半径，最后需要输入分割位置的种子点，种子点可以是多个。调用子模块有四个，分别是滚球膨胀、滚球腐蚀、区域联通、中值平滑。算法的处理逻辑如下：

If(滚球半径)

{

阈值滚球腐蚀；

If(中值平滑)

{

中值平滑；

}

区域联通；

滚球膨胀；

}

else

阈值区域联通；

四个子模块分别对应四个滤波器，具体对应关系如下：

滚球腐蚀：FastBinaryErodeImageFilter

滚球膨胀：FastBinaryDilateImageFilter

区域联通：ConnectedThresholdImageFilter

中值平滑：MedianImageFilter

其中中值平滑随半径的增大，计算时间增长最大，半径为5时最高需用时150秒。

1. **典型算法GPU加速工作**

在这个阶段主要是对四个典型算法做GPU算法移植工作，这个阶段工作一个是算法的加速比，另外一个就是算法的正确性。四个算法各有不同，算法各异，为达到较高的加速比，做了大量的工作，在这个过程中有的GPU并行算法被推倒重来好几次，反复打磨才推出最优版本。现在对每一个算法做简单说明工作。

1. FastBinaryErodeImageFilter

这个算法是公司自研算法，不是itk库的原始函数。算法逻辑很简单，就是滚球半径内的边缘点被腐蚀。也就是说，每一个值为1的点的滚球半径内如果有0，则该点被置为0。由于值为1的点比较少，所以直接按算法逻辑实现就行了，计算量不是很大。算法加速比随半径不同在90-327之间变化。

1. FastBinaryDilateImageFilter

这个算法也是公司自研算法，不是itk库的原始函数。算法逻辑很简单，就是滚球半径内的边缘点被膨胀。也就是说，每一个值为0的点的滚球半径内如果有1，则该点被置为1。但由于值为0的点很多，如果按腐蚀的算法思路来直接处理，计算效率会低很多，所以为提高速度，将公司自研算法设计相应的GPU并行算法，达到了较高加速比。算法思路就是先标记边缘点，然后对边缘点进行膨胀计算，大大加快了算法速度。算法加速比随半径不同在44-81之间变化。

1. ConnectedThresholdImageFilter

这是区域连通算法，也是肝分割的核心算法。由于需要做很多迭代工作，所以该算法的并行度不是很高。GPU并行算法设计的时候也是遇到很多困难。后来的处理流程是先把置为1的位置筛选出来，区域连通就是把这些值为1的点连通起来。先做XY平面的连通，先求X轴方向的基本连通关系和Y轴方向的基本连通关系，再通过XY轴交叉点来求XY平面的连通关系。然后再用X轴和Y轴的基本连通关系来检验XY平面连通关系的结果。反复迭代下去直到XY平面连通关系不再改变为止。同理把XY平面连通后的结果再和Z轴方向做连通，也是要求Z轴方向的基本连通关系，然后用Z轴方向的基本连通关系和XY平面的连通关系来检验XYZ的连通关系，反复迭代以后得到最终结果。所谓的连通关系其实就是如果一个集合里的点是连通的，那样所有的点都指向这个集合里坐标位置最小的点。这样就把所有的点通过连通关系分了类，最后再用种子点来确定哪些类该输出。所有输出的类合并在一起统一置相应的坐标位置的值为1作为输出。这个算法的加速比在36倍左右。

1. MedianImageFilter

中值滤波的算法逻辑很简单，但计算量巨大。中值滤波就是把一个立方体内点排序，求中间那个值作为输出。Image里的每个点都要求半径范围内的中值，由于还涉及到排序，所以计算量巨大。GPU并行的中值滤波算法对算法做了优化处理可以达到很高的加速比。该算法的加速比随半径不同在80-90之间变化。注意，中值滤波算法在itk的内部实现是并行的，也就是说如果算单卡对单核的加速比，达到了惊人的320-360倍加速。

1. **GPU算法的产品化融合**

GPU算法完成以后就该将GPU算法融合进现有的产品中。由于GPU每次运行的时候初始化也会消耗不少时间，所以我专门定义了一个类来做GPU的初始化和显存管理工作。通过实现这个类，可以不用每次启动GPU算法都开辟内存，大大减小了调用GPU函数的时间开销。但由于现有产品的模式是数据在CPU内存中，需要GPU加速的时候把数据从CPU拷贝到GPU中去，GPU计算完成后再把数据从GPU拷贝到CPU中，所以这个进出的时间是很难节省的。

通过对每一次算法调用耗时的分析，发现GPU和CPU之间拷贝的时间并不是很多，而是每次调用一个封装的算法耗时太多。每次调用封装算法，都是VTK到ITK，计算完成后再ITK到VTK，大量的时间就消耗在这些无意义的格式转换上去了。

为了优化算法调用流程，我将整个肝分割算法封装到一个算法接口中，将四次算法调用改为了一次算法调用。GPU和CPU之间的内存拷贝也只需要来回各一次。每次算法调用加上GPU计算时间都能控制在500毫秒以内。但整个程序框架还是要消耗大量的时间在消息以及界面的处理上，最终在点下计算按钮到计算完成显示结果，时间需要1200毫秒。实际GPU计算时间在100毫秒左右。

1. **GPU的算法创新**

通过计算流程可以发现，当腐蚀处理完成以后会进行中值滤波处理，这其实是一个二值中值滤波，就是说滤波器的输入就是0或者1。这样就不用生搬硬套中值滤波算法了，可以设计一个专门的优化的二值中值滤波算法。将二值中值滤波算法升级为GPU优化的二值中值滤波算法，加速比超乎想象。随半径的不同，加速比从269到1416。由于ITK的中值滤波是多核并行的，如果算单卡对单核的加速比，那就要到1076至5664。这么高的加速比体现的就是算法优化的威力。以前如果选取平滑半径为5，要等2-3分钟，现在100毫秒就计算完成了。所以不管界面上选择什么半径，都能把GPU计算时间控制在200毫秒以内，我直接就把进度条取消了，结果只要1秒就能计算完成了。

1. **测试案例**

为了验证算法结果，特别是要确定GPU的计算结果和CPU完全一样，我设计了测试方法，按这个测试方法完成了几个测试案例检验对比GPU和CPU的计算结果完全正确。测试程序也可以提交测试人员进行测试。

具体的测试方法为，确定一组输入参数，包括两个阈值两个半径若干种子点，然后开始计算。最后的输出结果缓冲区做一个MD5 hash，对比MD5值就能知道计算结果是否完全一样。这样测试，能重现结果，能快速检验结果，而且能确保CPU和GPU的计算结果完全一样。只有这样，才能保证GPU程序的正确性，才能将GPU程序融合进公司的产品中。

1. **附件**

我将GPU算法都打包到一个测试程序中，运行完毕即可得到GPU运行正确与否以及GPU程序的加速比。附件中包括开发中的测试程序输出结果和最终测试结果。

1. **Bug修正**

yRegionGrowingAlg算法调用存在处理流程上的混乱。yRegionGrowingAlg内部调用了ConnectedThresholdImageFilter类，如果是在算法的开始阶段就调用，输入两个阈值和种子点做区域连通计算是没问题的。如果是在分割算法处理过程的中间调用该类，如果再传阈值进去，就没必要了，而且在初始化输入inputimage时，算法还有bug，并不能保证前一步分割数据正确输入，从而导致计算错误。为优化算法流程和保持正确性，可以将前期分割数据直接作为ConnectedThresholdImageFilter的输入，将上下阈值都置为1，就能让ConnectedThresholdImageFilter类处理中间的分割图像了。