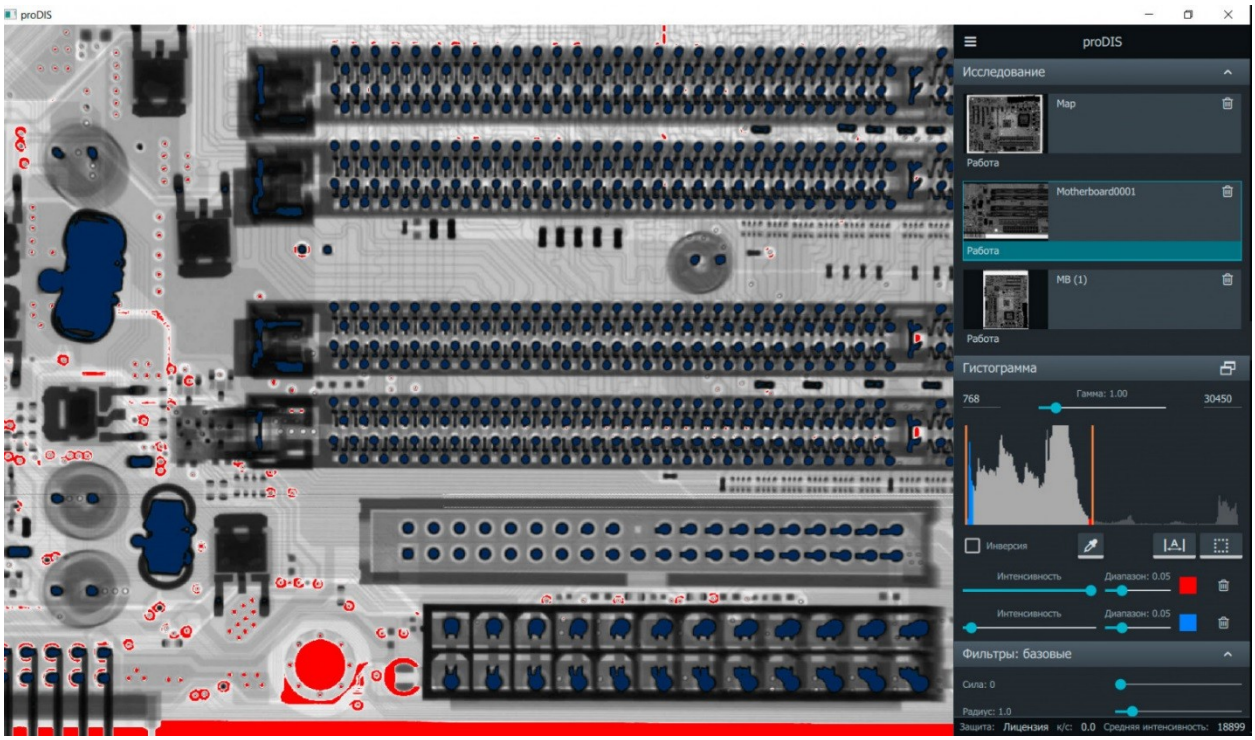
Цифровой рентген: прогулка по Эльбрусу

Привет Хабр! В прошлый раз писал про проект по рентгеновской инспекции печатных плат. Сейчас мы сильно продвинулись, есть рабочий прототип софта плюс “потыкали палочкой” в Эльбрус. Про этот опыт я и хочу рассказать.



Интро

На старте проекта нам удалось найти дополнительное финансирование, основным условием была полная кроссплатформенность ПО, в том числе поддержка отечественных процессоров. На тот момент наиболее производительным вариантом для десктоп машин был Эльбрус 8С (пока он им и остается, 8СВ еще вроде не вышел). Мы купили две станции «Эльбрус 801-РС» напрямую от МЦСТ. Сейчас их стоимость указана на сайте, год назад были чуть дороже.

Из курьезных мелочей, с которыми столкнулись при закупке – бумажные заявки на поставку. Сразу советую заключить договор NDA, это даст доступ к свежим инструментам разработчика (оформление около месяца). Приехали машины быстро, по комплектации – есть проблема с кастомизацией на стороне МЦСТ. Проще докупить и поставить самим нужные видеокарты или периферию. Перечень проверенного оборудования/чипов карт пока есть только в багтрекере МЦСТ, хотя стоило бы опубликовать список на вики ресурсе по Эльбрусам.

Операционные системы

По ситуации на начало 2020го года – при заказе можно выбрать предустановленную ОС: оригинальную Эльбрус или партнерскую (Альт, Астра). В нашем проекте используется Астра Linux. Она хоть и была доступна официально, но по факту – не получилось заказать за адекватное время. Ставили сами с лайв образа отдельной лицензией.

Выбор той или иной ОС – дело вкуса, мне понравилось окружение Fly в Астре, субъективно оно шустрее. С другой стороны, пакетная база Альта шире, они теснее сотрудничают с МЦСТ и набор дистрибутивов интереснее.

Пример – не существовал USB лайв образ Астры под Эльбрус, а для Альта их было аж несколько штук. Бэкап раздела Астры работал только из-под лайв образа Альта.

Сама ОС Эльбрус изначально разрабатывалась для демо процессора заказчикам. Мы начинали с версии 4.0, сейчас рабочая 5.0rc2 – стало лучше, но все же она «сыровата» для конечного пользователя. Нужна для отладки или получения максимума от VLIW архитектуры. Именно на этой ОС производительность при обработке изображений была максимальной.

UPD: сейчас вышла версия ОС Эльбрус 6.0. Там заявлены C++20 и свежее ядро Linux, но так как мы работаем на Астра — обновления еще не добрались.

Архитектура

Пока только C++, 14-ый стандарт, про CUDA и Vulkan лучше не думать, а вот OpenGL на AMD видеокартах работает нормально. Важно – OpenGL не старше 3.1, QT 5.11.

По остальным ЯП – возможно кто-то поделится своим опытом в комментариях. Знаю, что идет работа в «закрытых» конторах, в основном по обработке видеопотока и изображений. Понемногу народ осваивает отечественные процессоры.

Теперь слово архитектору проекта, **Максиму Титову** (titovmaxim, Unicore Solutions)
Наша цель – сделать хороший софт для работы с рентгенографическими системами. Поэтому начали с главного – подключения железа и тракта обработки изображений. Сильно волновались. TLDR: все прошло гладко.

Главная часть системы – детектор рентгеновского излучения. Подключается по Ethernet и забивает канал 1Гбит под завязку, протокол GigE Vision. Коммерческие библиотеки под Эльбрус отсутствуют, библиотеки с открытым исходным кодом (например Aravis) не подходят по скорости и качеству, так что писали свою реализацию.

Столкнулись с тем, что встроенная сетевая карта материнской платы Эльбруса сильно нагружает процессор, и 1ГБит просто так не получить, теряются кадры. Пришлось дописывать дополнительный быстрый промежуточный буфер, чтобы разгрузить поток приема. Обрабатываем все потом в отдельных потоках. Все заработало, канал забивается полностью, потерь кадров нет. При установке на Эльбрус дискретной карты проблем становится заметно меньше.

Предварительная обработка изображений в реальном времени реализована на OpenGL, т.к. это единственное средство, доступное нам на всех платформах. В нее входит применение калибровки и коррекция битых пикселей детектора, обработка гистограммы, гамма коррекция, повышение резкости и подавление шума, раскраска полутонов, геометрические преобразования. Интересно было написать быструю оценку гистограммы изображения на OpenGL, а не CUDA, задача не совсем типичная.

Итого на обработку одного кадра 3000x3000 16 бит на самой слабой видеокарте (AMD R5) на Эльбрусе уходит ~33 мс. Очень даже неплохо, при условии независимости от ОС и полной разгрузки ЦП для задач сложной пост-обработки. Уже доступны более мощные видеокарты, так что здесь мы спокойны. К примеру, на x86 с GeForce RTX 2070 Max-Q мы получаем стабильные ~2мс, ждем подобного на Эльбрус.

Подключение прочих компонентов – рентгеновских генераторов и мехатроники по RS232, Web камер по UVC, прошло без проблем.

В проекте мы используем Qt 5.11 и QML для интерфейса. Здесь все прошло неожиданно гладко. Работает, заводится "из-коробки", основные пакеты в Астра есть. Ждем, когда будут доступны обновления, поскольку в 5.11 есть «шероховатости» и баги.

Немного о том, почему мы опасались C++ 14. Активное совместное использование C++ и QML часто вызывает сложности, поэтому мы решили использовать в проекте свою библиотеку Flow.

Библиотека Flow

Среди плюсов библиотеки — декларативность, сокращение boilerplate кода и ошибок. Помогает использовать ФП подход на C++, в частности есть гибкое объединение и композиция функций, ленивость, кеширование, потокобезопасность и фоновое выполнения в других потоках. Последнее, кстати, актуально для OpenGL, который не очень «умеет в многопоточность». Все это в динамике с оповещением об изменениях (никаких update) отстыковки/пристыковки веток, действия (эффекты в смысле ФП), интегрированы с контекстами Qt. Из приятного – автоматический контроль времени жизни объектов и подписок без ручных subscribe/unsubscribe и беспокойств, кто кого переживёт, все само :) Немного похоже на ReactiveX, но тут больше про состояния, а не про потоки данных.

Там же живет своя мета-система (в C++ рефлексию не завезли), используем вместо QMetaObject. Писать меньше, лучше интегрируется с QML (примерно, как WPF с C#), можно свободно перемещаться по дереву, в частности работать из QML с QVector в середине дерева как с моделью с умным diff'ом (без написания QAbstractItemModel), автоматическая сериализация/десериализация любого объекта одной командой и пр.

Библиотека изначально была рассчитана минимум на C++ 17. При переходе на C++ 14 мы, естественным образом, потеряли вывод типов и сейчас приходится прописывать большинство шаблонных параметров руками. Однако прогресс МЦСТ впечатляет, ждем новых компиляторов. Была пара особенностей компилятора Эльбрус, не замеченных ранее на GCC и MSVC. Не всегда понимаются auto параметры в лямбдах. Не умеет перезахватывать this во вложенные лямбды. Но это легко правится, в остальном все

компилируется и работает без проблем. Да, ошибки компилятора на русском языке немного непривычны ;)

Вследствие особенностей архитектуры на Эльбрус не рекомендуется использовать исключения. Поэтому мы сразу отказались от них в нагруженных частях. Однако достаточно широко используем их на верхнем уровне, где операции единичные. Все работает как положено, зря боялись. Да, пока не завезли -fnon-call-exceptions для удобного отлова ошибок уровня системы и их обработки на месте возникновения.

Приятно, что можно большую часть кода писать "дома" на Linux под x86 и потом просто собирать на удаленной машине Эльбруса. При некоторой сноровке проблем с совместимостью кода почти не возникает.

Производительность OpenCV

Наиболее критичные вещи по обработке видеопотока вынесены на видеокарту, для «неторопливой» пост-обработки изображений используем OpenCV 3.2. Этот пакет портирован на Эльбрус, но есть “нюанс” – производительность сильно зависит от версии пакета для конкретной ОС. См. таблицу по сравнению производительности пакетов OpenCV на Эльбрус 8С (1300 МГц) и Intel core i7 (2600 МГц) под разными ОС/сборками openCV:

Таблица сравнения openCV Эльбрус vs Intel i7

Время выполнения операции, мс		**ОС Эльбрус 5.0rc2/ Эльбрус-8С/ OpenCV 3.2**	**ОС Астра Ленинград 8.1/ Эльбрус-8С/ OpenCV 3.2**	**ОС Астра Смоленск 1.6/ Intel Core i7 7700/ OpenCV 3.2**	**ОС Windows 10/ Intel Core i7 9750H/ OpenCV 3.2**	**ОС Windows 10/ Intel Core i7 9750H/ OpenCV 4.4**
Свертка	ядро 5x5, 3000x3000, 16S	35	334	99,7	94	105,9
Свертка	ядро 5x5, 3000x3000, 16U	244	280	-	98	106,5
Свертка	ядро 5x5, 3000x3000, 32F	32	271	23,9	24	11,4
Гауссово размытие	ядро 5x5, 3000x3000, 16S	15,3	257	36,3	35	5,7
Гауссово размытие	ядро 5x5, 3000x3000,	184	251	-	12,5	40

Время выполнения операции, мс		ОС Эльбрус 5.0rc2/ Эльбрус-8С/ OpenCV 3.2	ОС Астра Ленинград 8.1/ Эльбрус-8С/ OpenCV 3.2	ОС Астра Смоленск 1.6/ Intel Core i7 7700/ OpenCV 3.2	ОС Windows 10/ Intel Core i7 9750H/ OpenCV 3.2	ОС Windows 10/ Intel Core i7 9750H/ OpenCV 4.4
	16U					
Гауссово размытие	ядро 5x5, 3000x3000, 32F	14,5	222	8,1	7,7	6,2

Производительность OpenCV на Эльбрусах напрямую завязана на низкоуровневые EML библиотеки (см. руководство по программированию МЦСТ, они оптимизированы под VLIW архитектуру). А пакеты EML зависят уже от дистрибутива ОС. В используемой нами Астре свежая сборка до сих пор не появилась, возможно она есть в Альт Линукс. Кто игрался – напишите в комментариях.

Если говорить про рутину – свертку изображений, то производительность может быть в 2 раза лучше (16S) по сравнению с i7, а может быть и в 2 раза хуже (32F). На сборке ОС с неоптимизированной библиотекой OpenCV проигрыш в производительности **до 20 раз**. И да, с 16U у Эльбруса пока все плохо.

Резюме

Жить на Эльбрусе можно. Постепенно МЦСТ и партнеры (Альт, Астра) допиливают ПО, улучшают сервис, снижают цены. Это радует.

С другой стороны – их ресурсы ограничены, чего-то нужного именно вам можно ждать полгода.

Возможно правительство станет более активно продвигать отечественные процессоры и стимулировать их коммерческую эксплуатацию, посмотрим. Мы в любом случае к этому готовы.