



XII международная конференция
CEE-SEC(R) / РАЗРАБОТКА ПО

28 - 29 октября, Москва



Технология предметно-ориентированного программирования для неоднородных многоядерных платформ

Борис Седов, Юрий Шейнин, Алексей Сыщиков

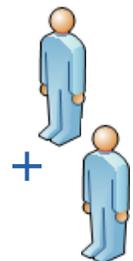
boris.sedov@guap.ru, sheynin@aanet.ru

Санкт-Петербургский государственный
университет аэрокосмического приборостроения



Зачем нужна такая технология?

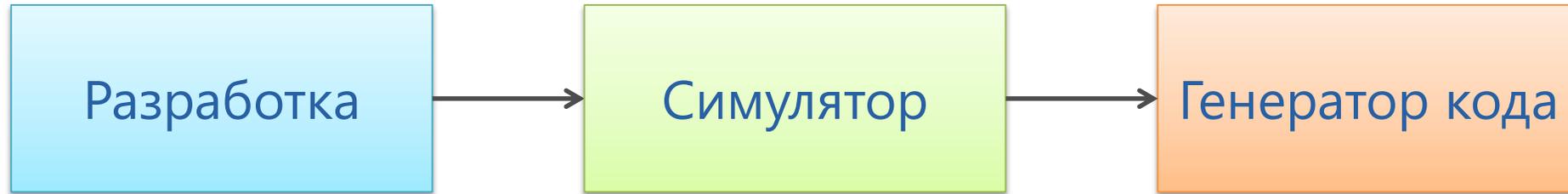
1. Требуется разработчик "два в одном":



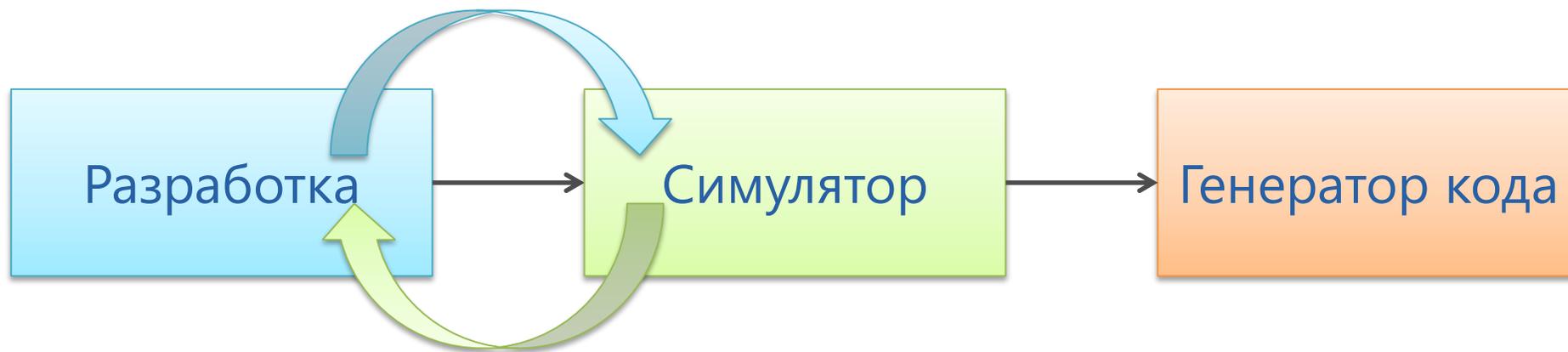
квалифицированный специалист в предметной области
+ квалифицированный программист

2. Сложность разработки параллельного встраиваемого ПО
3. Неоднородность встраиваемых систем
4. Аппаратные платформы быстро устаревают, но решаемые на них задачи и алгоритмы устаревают редко
5. Для достижения требований необходима адаптация алгоритмов под платформы и платформ под алгоритмы

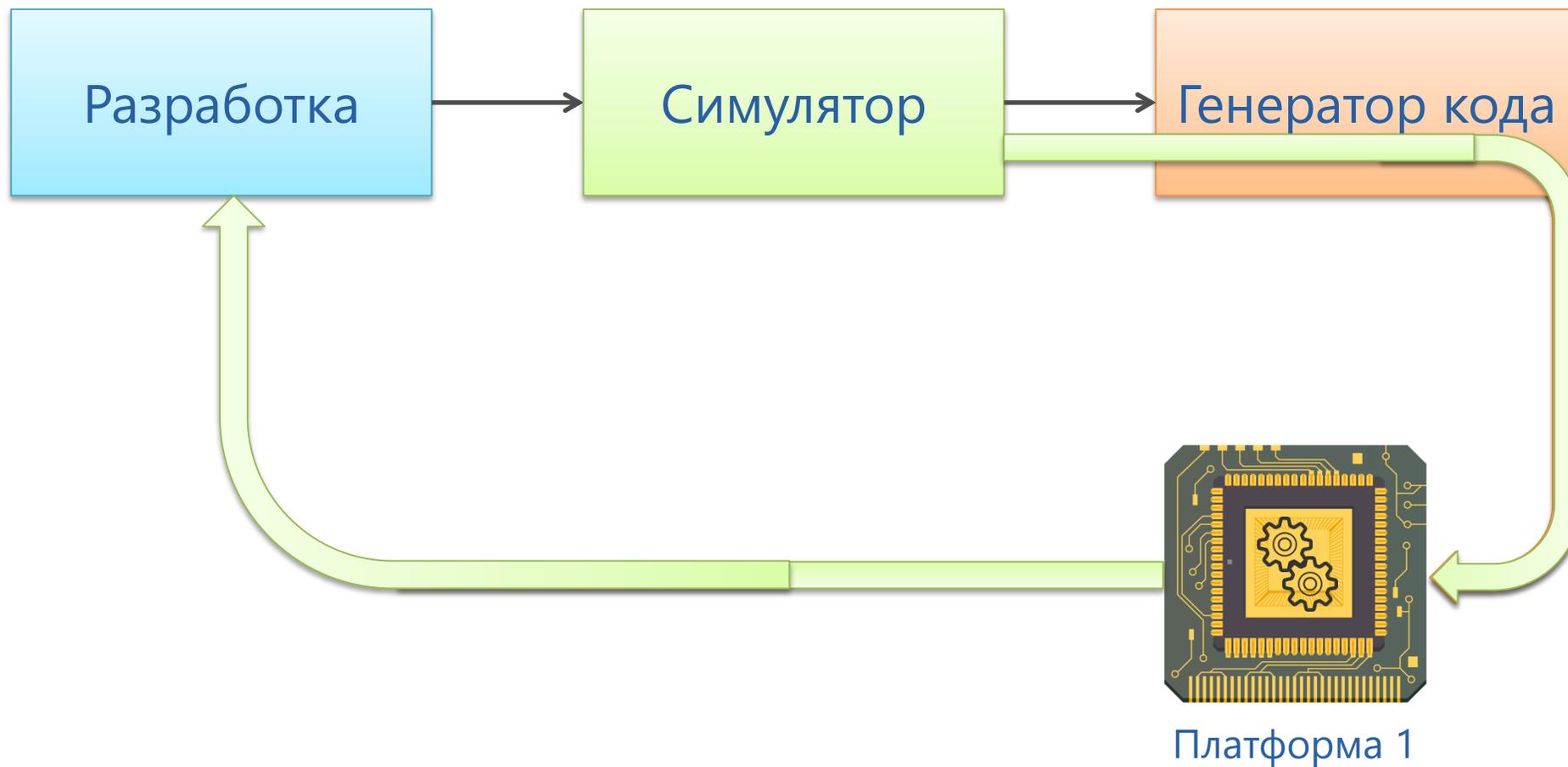
Жизненный цикл программы в рамках технологии



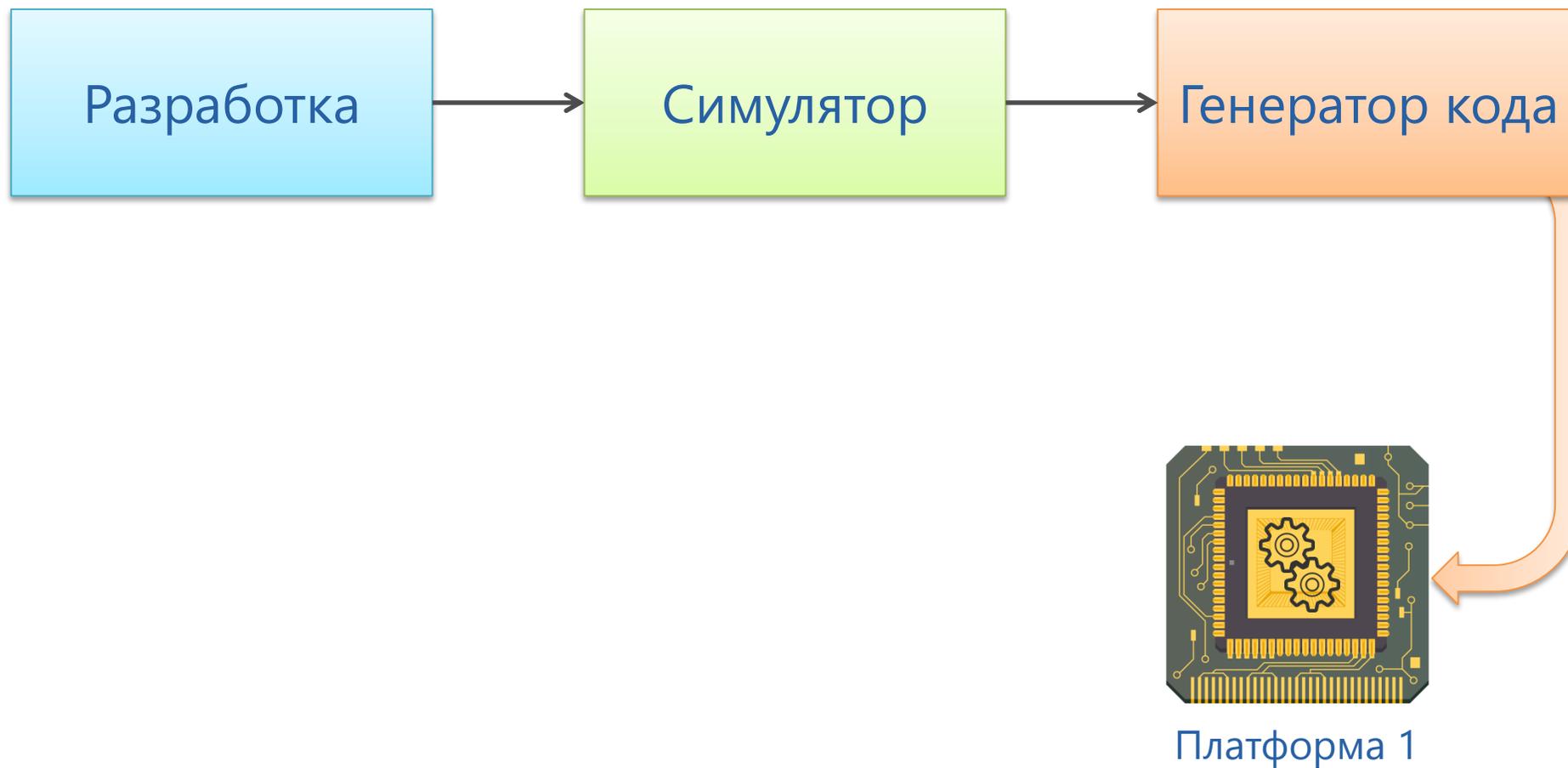
Жизненный цикл программы в рамках технологии



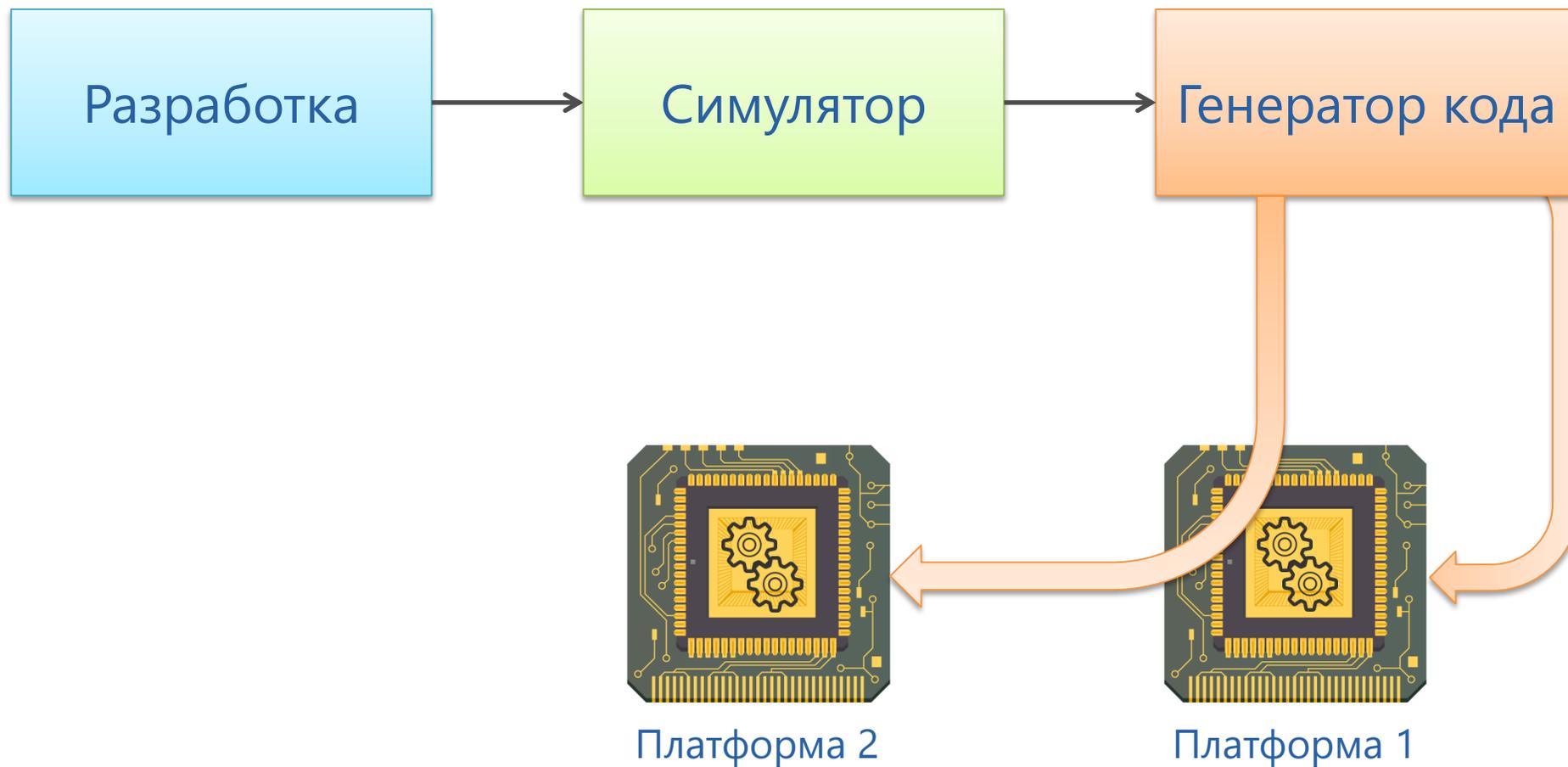
Жизненный цикл программы в рамках технологии



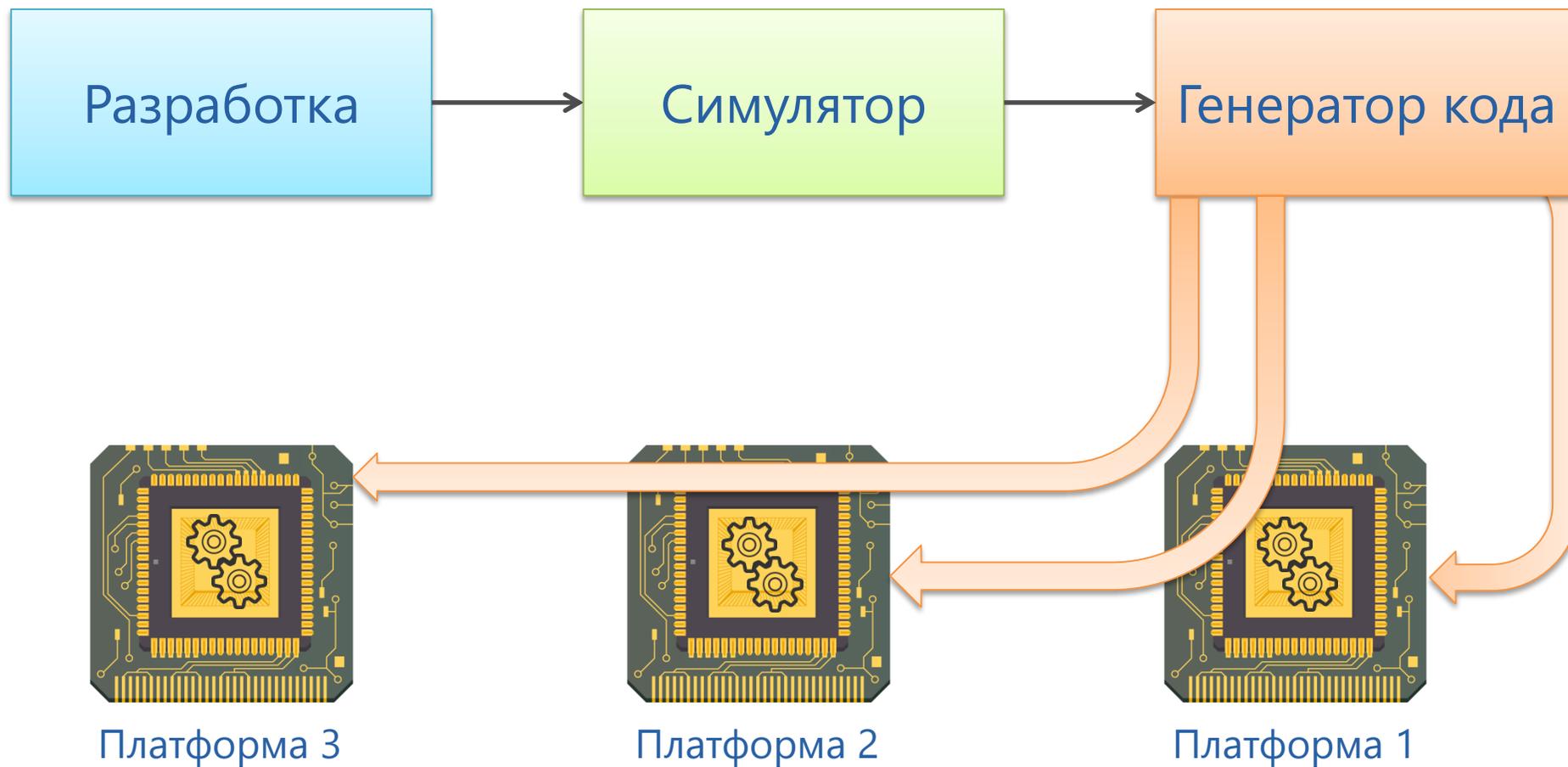
Жизненный цикл программы в рамках технологии



Жизненный цикл программы в рамках технологии



Жизненный цикл программы в рамках технологии





Специалист в предметной области

Детальная схема технологии



Детальная схема технологии



Детальная схема технологии

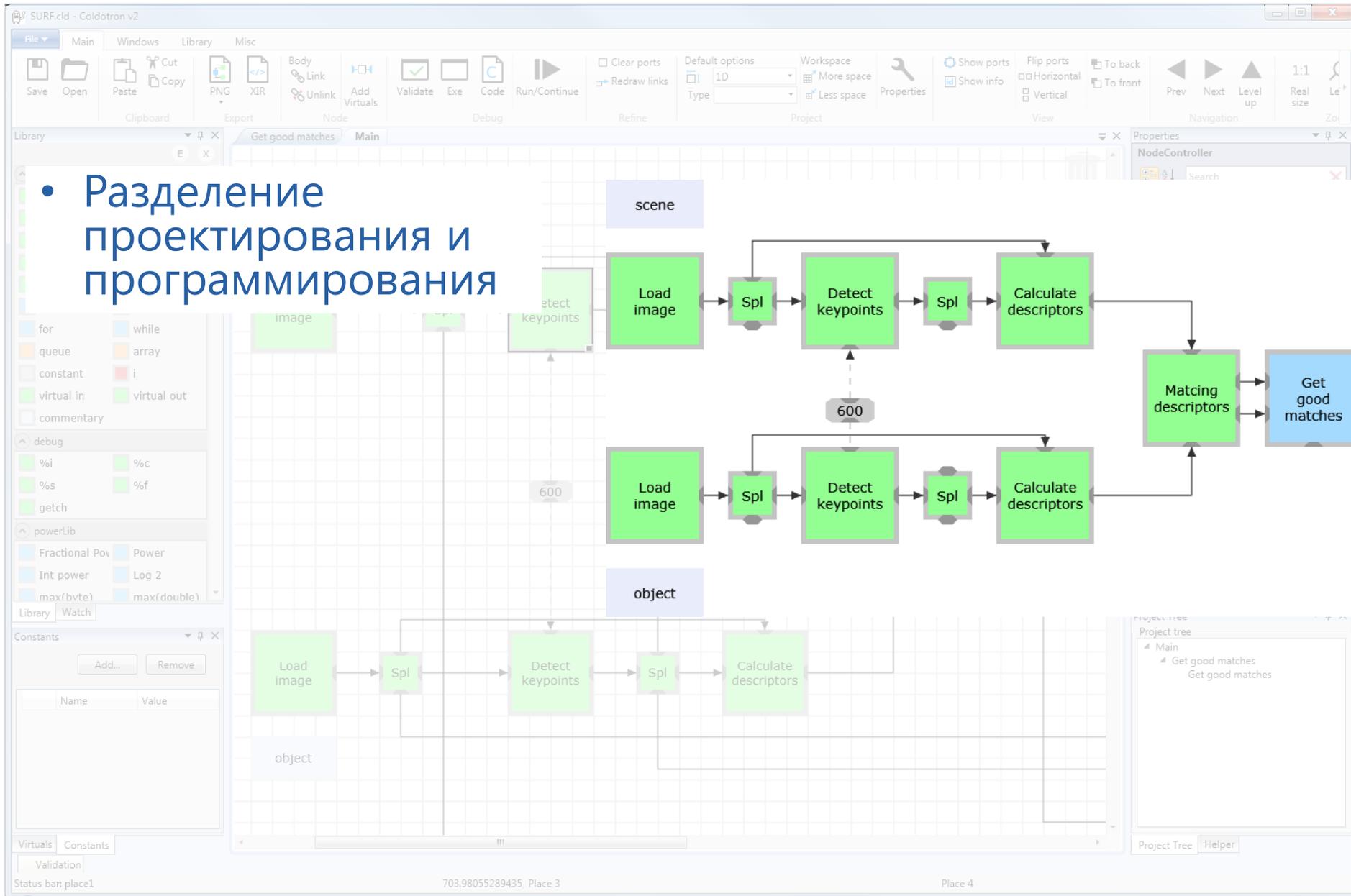


Детальная схема технологии



Графический язык программирования VPL

- Разделение проектирования и программирования



Графический язык программирования VPL

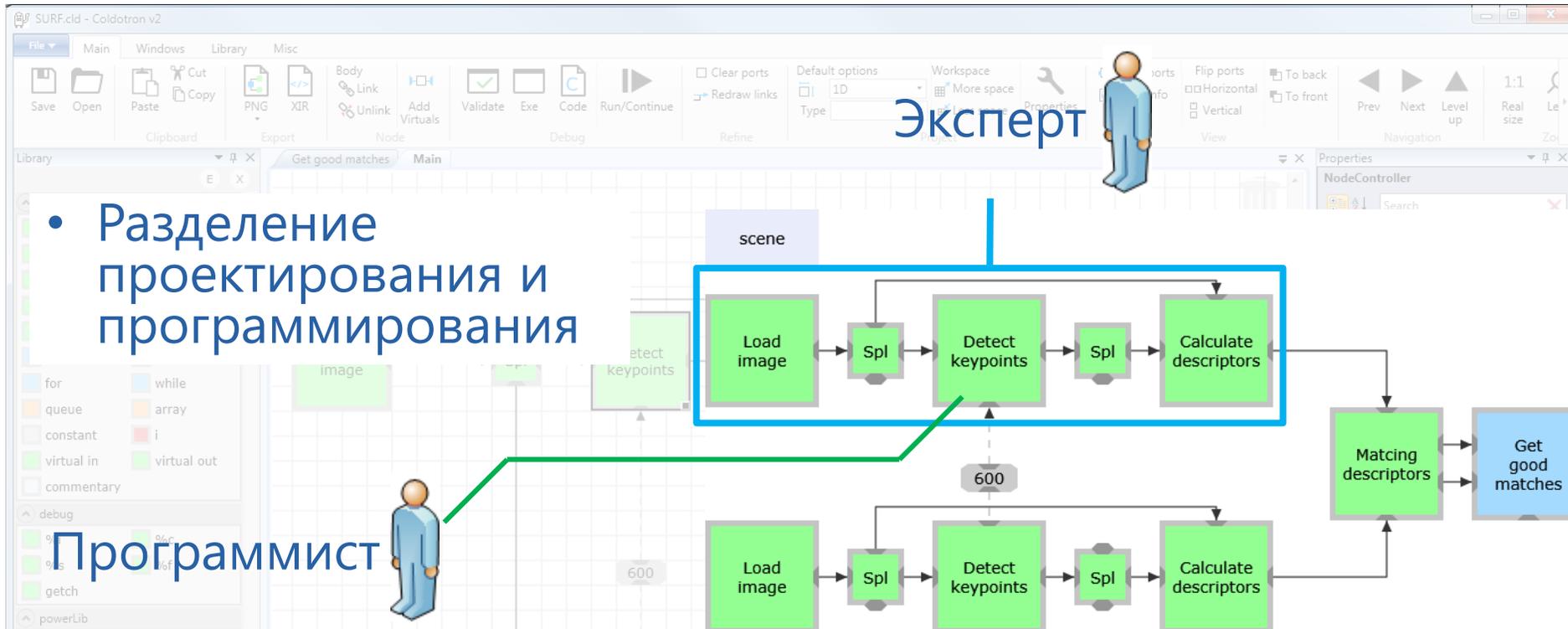
Эксперт

- Разделение проектирования и программирования

Программист

```
int dhCalcAirlight ( DataLink *in11, DataLink *in31, DataLink *out2: )  
{  
    memcpy(&p, in11->Data, sizeof(int*));  
    CImg<double>* brightestDarkPixels = (CImg<double>*)p;  
    memcpy(&p, in31->Data, sizeof(int*));  
    CImg<double>* data = (CImg<double>*)p;  
    float airLight[3] = { 0.0, 0.0, 0.0 };
```

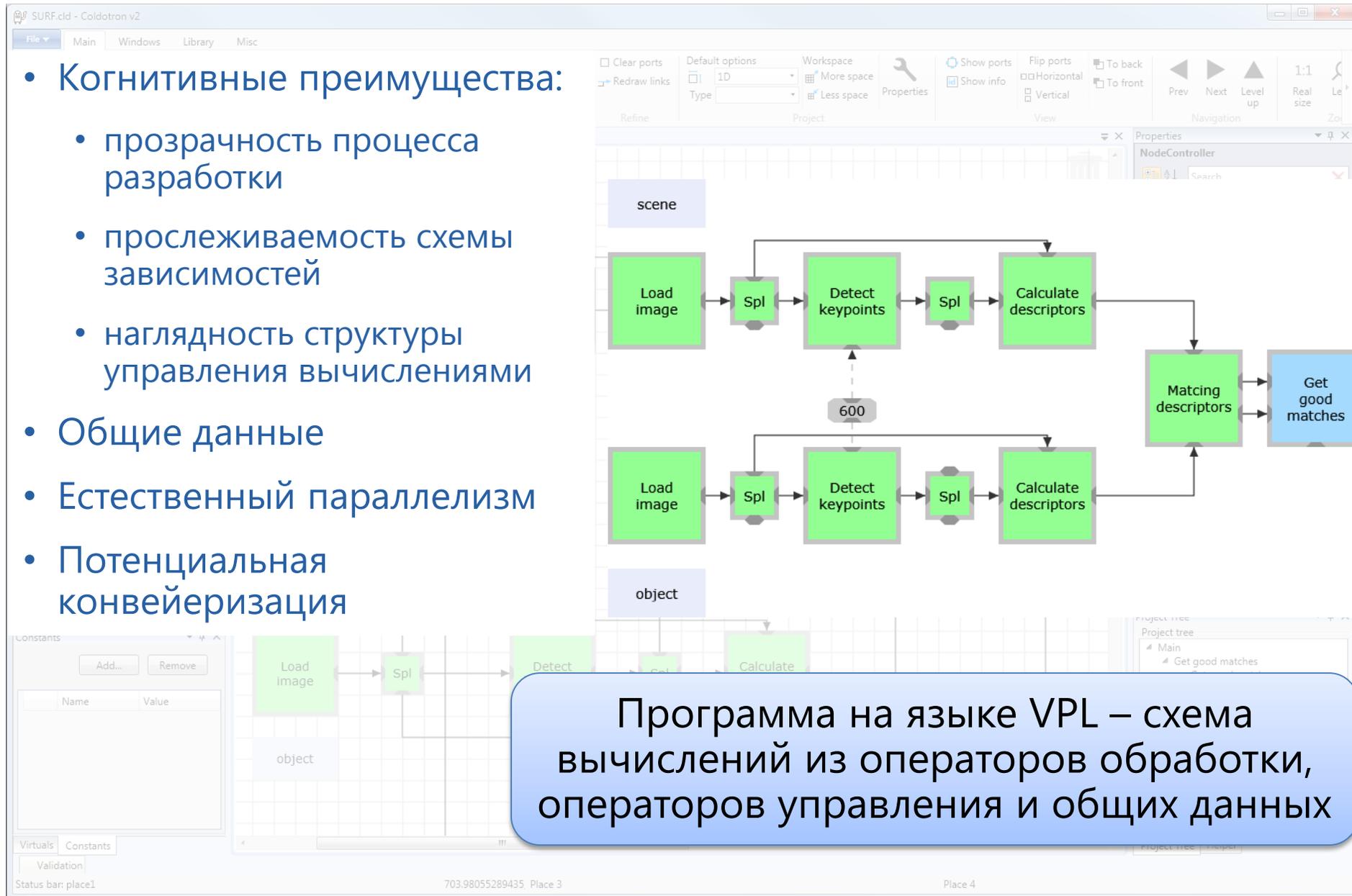
Графический язык программирования VPL



- Простота внесения изменений в структуру программы
- Отсутствие влияния программиста (кодера) на спроектированную схему
- Локальное проявление ошибок разработки
- Поддержка и сопровождение программ в течение жизненного цикла
- Уменьшение вероятности ошибок без потери читаемости программы
- Гибкость и лёгкость изменения на любом этапе разработки

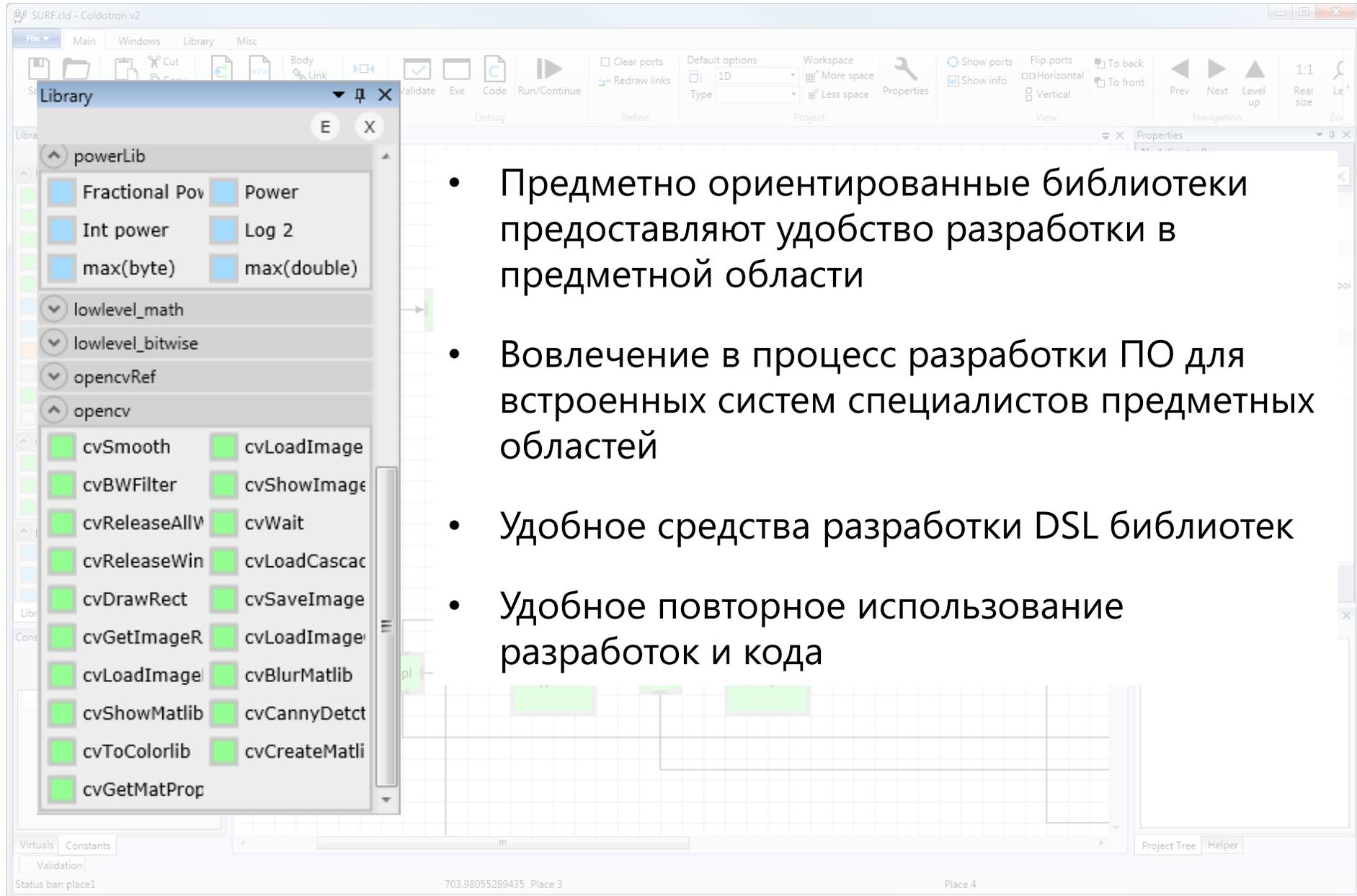
Графический язык программирования VPL

- Когнитивные преимущества:
 - прозрачность процесса разработки
 - прослеживаемость схемы зависимостей
 - наглядность структуры управления вычислениями
- Общие данные
- Естественный параллелизм
- Потенциальная конвейеризация



Программа на языке VPL – схема вычислений из операторов обработки, операторов управления и общих данных

Предметно-ориентированное программирование



- Предметно ориентированные библиотеки предоставляют удобство разработки в предметной области
- Вовлечение в процесс разработки ПО для встроенных систем специалистов предметных областей
- Удобные средства разработки DSL библиотек
- Удобное повторное использование разработок и кода

Формальная модель вычислений: Асинхронные Развивающиеся Процессы

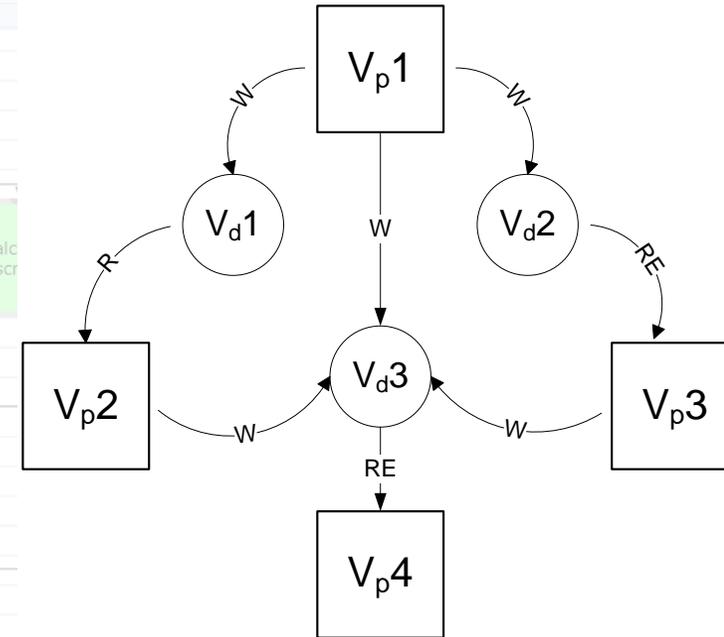
АРП-модель определяет

- синтаксис языка
- семантику объектов языка
- управляющие конструкции

АРП-модель предоставляет

- методы формальной верификации
- идентичность результатов при работе в разных окружениях
- возможности для переносимости вычислений
- представление динамики параллельного вычисления
- совмещение работы с общими данными и модели распределенной памяти

Схема программы – ориентированный граф



V_p – вершина-оператор,
 V_d – вершина-данное,
 $W/R/RE$ – дуги (связи) и их разметка

VIPE: визуальная среда проектирования

The screenshot shows the VIPE environment with the following components:

- Menu:** File, Main, Additional, Windows, Library, Misc.
- Toolbar:** Save, Open, Add Virtuals, Validate, Exe, Code, Run / Continue, Set print, Set getchar, Properties, Clear ports, Redraw links, Show ports, Show info, 1:1 Real size, Less, More, Autoscale, Undo, Redo.
- Library:** A list of components including 'i', 'vxData', 'virtual in', 'virtual out', 'commentary', 'glNode', 'debug', 'lowlevel_math', 'opencvRef', 'opencvVal', 'cvSmooth', 'cvLoadImage', 'cvBWFILTER', 'cvShowImage', 'cvReleaseAll...', 'cvWait', 'cvReleaseWi...', 'cvLoadCasca...', 'cvDrawRect', 'cvSaveImage', 'cvGetImage...', 'cvLoadImag...', 'cvLoadImag...', 'cvBlurMatlib'.
- Constants:**

Name	Value
ENABLE_EYES_D	0
ENABLE_EYES_D	0
DRAW_FRAMES_I	1
- Main Workspace:** A flowchart with nodes: 'Open Capture' (width, height), 'Face Recognizer Create' (Recognizer*, Height, Weight), 'Names', 'While' (loop), 'Face Recognition' (IF), 'Get Faces', 'IF faces > 0? - T', 'IF for all faces', 'IF eyes > 0 - T', 'IF Draw ey', 'IF For a', 'skip flag', 'SetupWiringX', 'frame number' (0).
- Properties Panel:** ControlNodeContro... with fields for Comment, Execution cost (ms), Iterations number, Parallel, Allocation, ID, Operation name, Permanent.
- Project Tree:** Главная, While, IF is frame%10==0-T, IF dont skip frame?-T, Face recognition, Get Faces, IF faces > 0? - T, F for all faces, IF eyes > 0 - T, IF Draw ey, F For a.

Интерактивные инструменты



Инструменты поддержки процесса разработки:

- валидация схемы

- проверка синтаксиса
- проверка типов
- корректность связей
- полнота схемы
- и т.д.

The screenshot shows the SURF development environment. On the left, there is a 'Main Library' with various components like 'functional', 'Spl', 'TG', 'Wr', 'if', 'for', 'que', 'cor', 'virt', 'cor', 'debu', '%i', '%s', 'get', 'pow', 'Fra', 'Int', 'ma', 'Library', and 'Constant'. The main workspace displays a block diagram with nodes like 'Mod', 'Spl', and 'I=' connected by arrows. A 'cond' node is highlighted with a red box. Below the diagram, a 'Validation' window shows the message 'Virtual 102 has 0 links'. On the right, a 'Watch' window displays the following data:

Name	Values
Link 425	[X]
Link 3651	[X]
Link 3652	[10]
array 446	[1][2][3][4][5][6][7][8][9][10]

Интерактивные инструменты



Инструменты поддержки процесса разработки:

- валидация схемы
- верификация (в работе)
- однозначность
- deadlocks
- livelocks
- завершаемость
- и т.д.

The screenshot shows the SURF development environment. On the left, a library of components is visible. The main workspace displays a state transition diagram with nodes for variables 'a' and 'b', and operations 'Mod', 'Spl', and 'I='. A red box highlights a 'cond' node. Below the diagram, a 'Validation' window shows the message 'Virtual 102 has 0 links'. On the right, a 'Watch' window displays the following data:

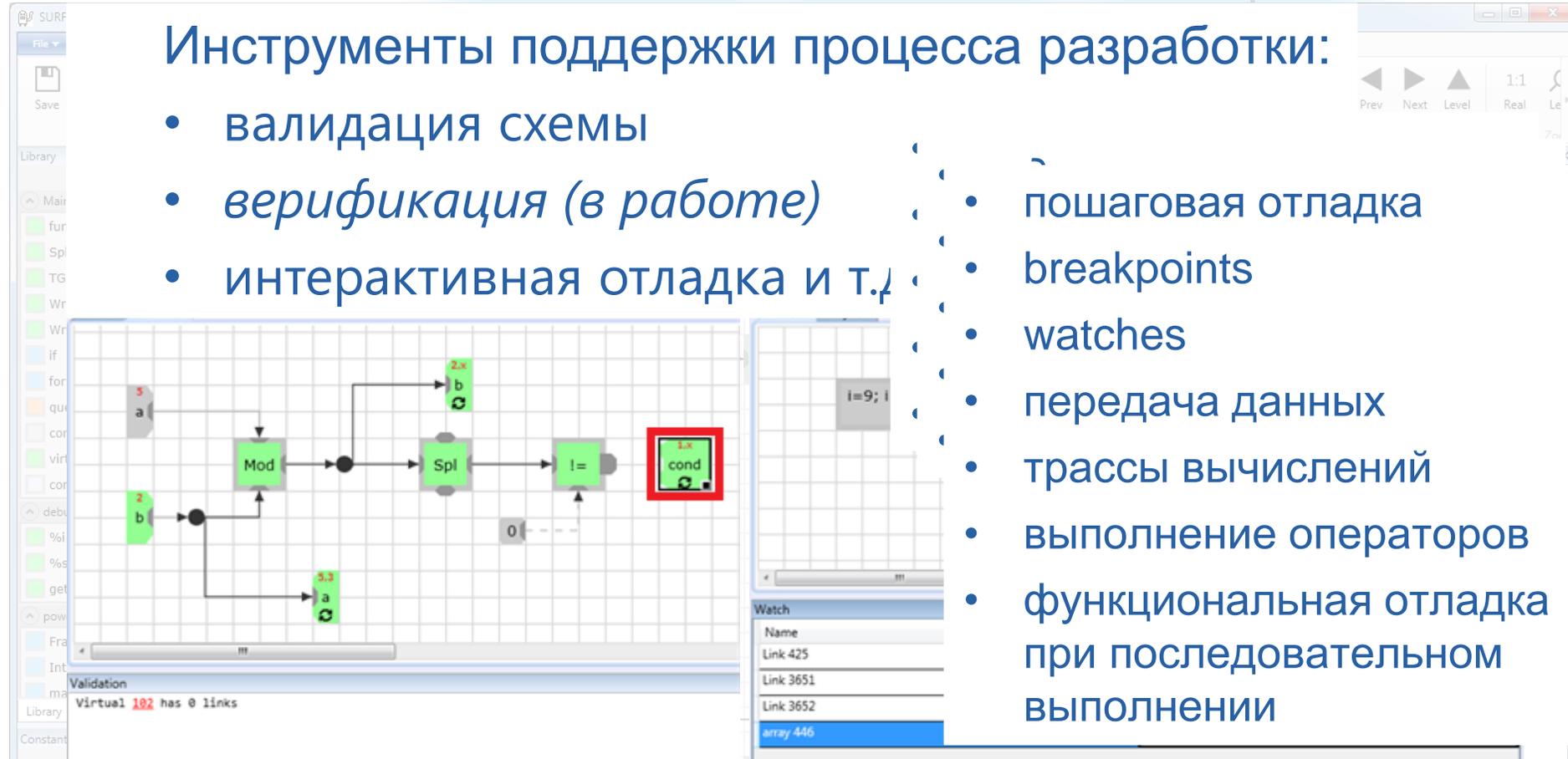
Name	Values
Link 425	[X]
Link 3651	[X]
Link 3652	[10]
array 446	[1][2][3][4][5][6][7][8][9][10]

Интерактивные инструменты



Инструменты поддержки процесса разработки:

- валидация схемы
- *верификация (в работе)*
- интерактивная отладка и т.д.
- пошаговая отладка
- breakpoints
- watches
- передача данных
- трассы вычислений
- выполнение операторов
- функциональная отладка при последовательном выполнении



Средства проектирования параллельных программ



Инструмент анализа «визуальный профилировщик»

Поиск критических участков (hotspots) в программе

Профилерование

Пороговое значение (%) Относительный % Критические участки Абсолютный %

Имя	Подтип	Ср. кол-во итер.	ID	%	Среднее время
Process image N times	F	20	3137	71,4 %	8,004 s
Detect ROIs	C		1432	3,31 %	371,5 ms
Detect regions	IF		1781	3,31 %	371,5 ms
Postprocess ima	IF		2696	3,08 %	345,7 ms
Enhance eac	F	51.6	2741	3,02 %	338,7 ms
Enhance	T		2939	0,06 %	6,564 ms
Locate all RC	T		2738	0,06 %	7,031 ms
Spl	T		3026	0 %	0 us
Spl	T		2896	0 %	0 us

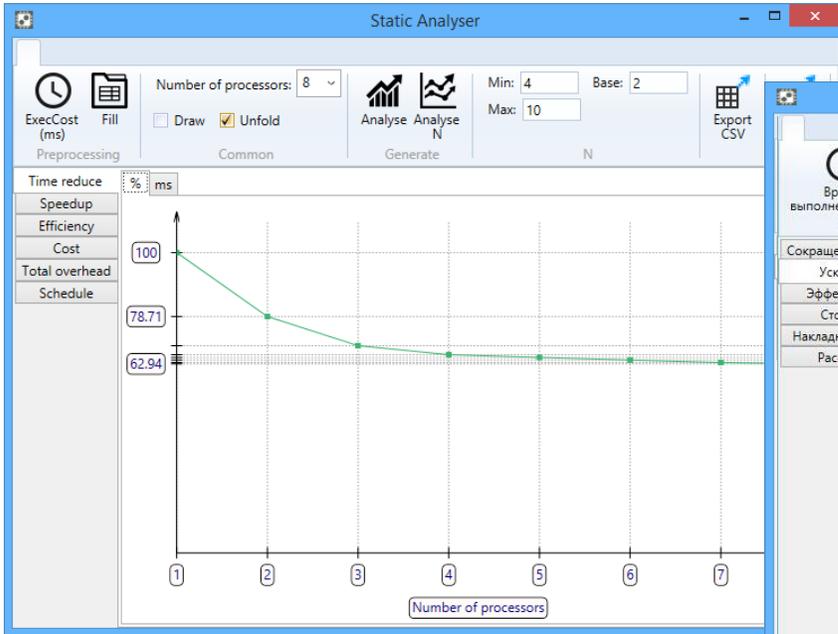
Общее время: 11,21 s

Различные режимы работы:

- Абсолютные времена выполнения операторов схемы
- Относительные времена согласно структуре вложенностей
- Критические участки с фильтрацией малых величин

Инструмент статического анализа

Быстрая оценка работы ПО на модели многоядерной платформы

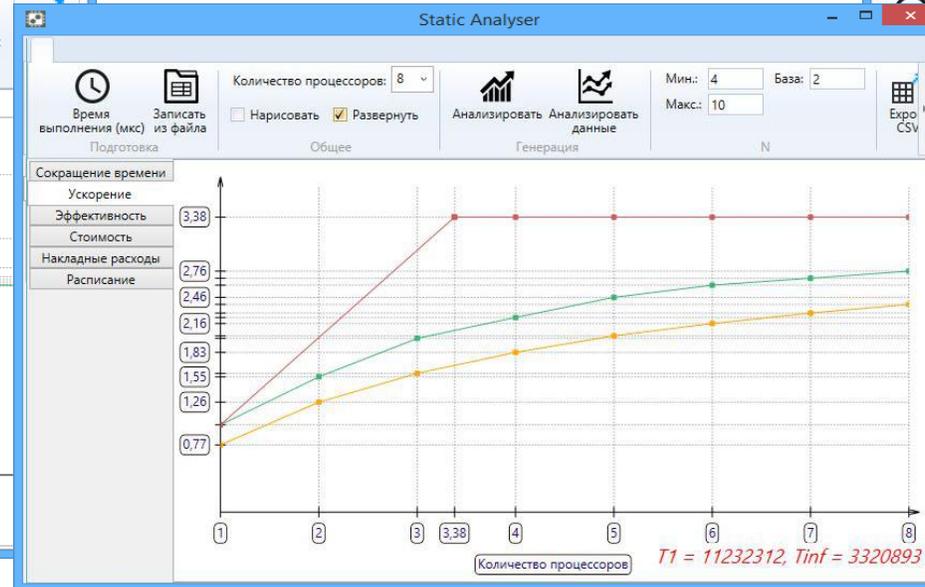


Время

$$R = \frac{T_n}{T_1} * 100\%$$

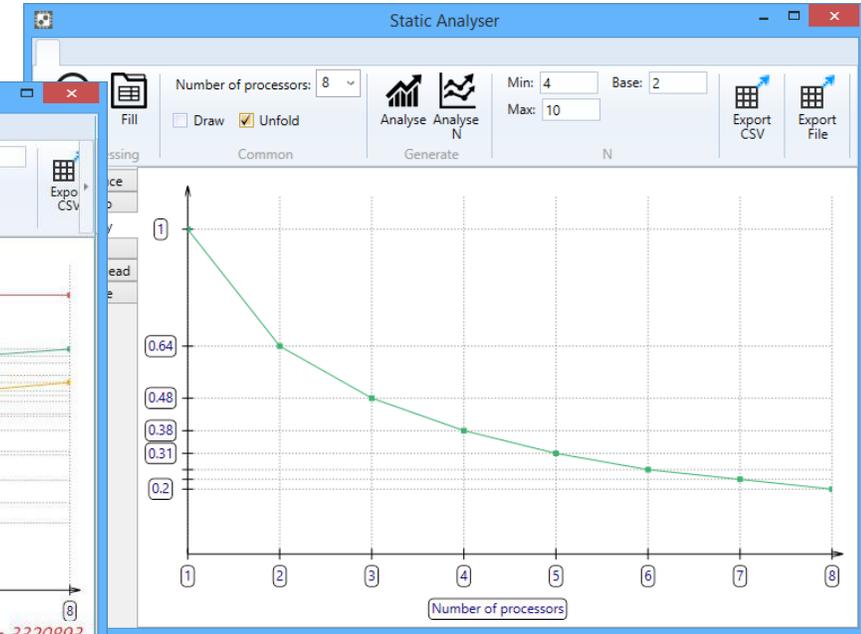
T_1 – время выполнения программы на 1 процессоре

T_n – время выполнения программы на N процессорах



Ускорение

$$S = \frac{T_1}{T_n}$$



Коэффициент эффективности

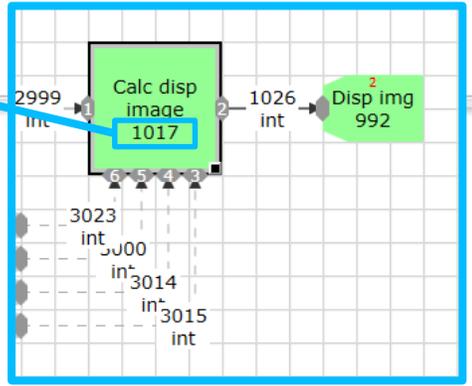
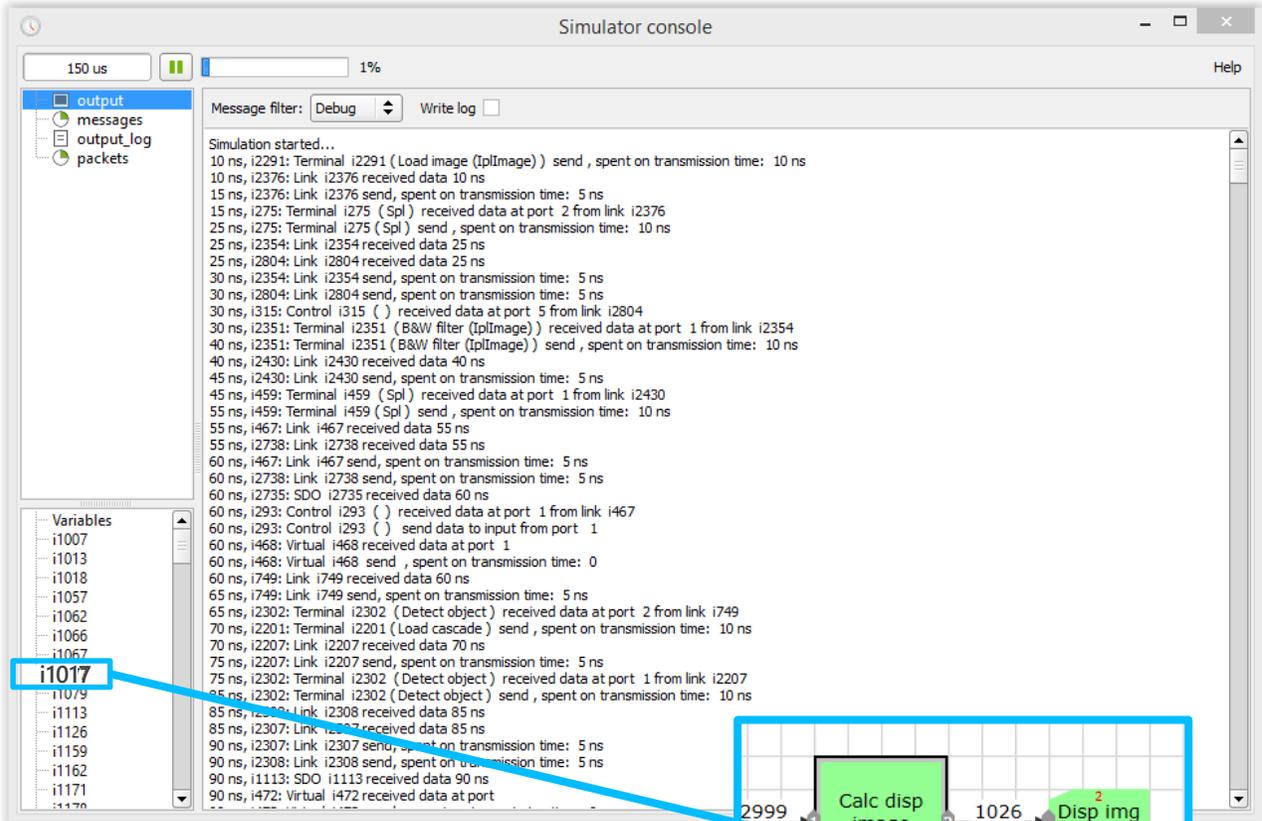
$$E = \frac{T_1}{N * T_n}$$

N – количество процессоров

VPL-симулятор

Позволяет оценить:

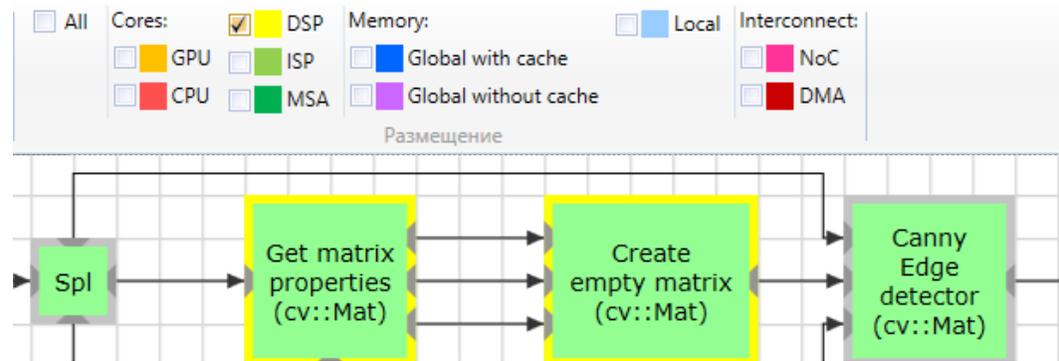
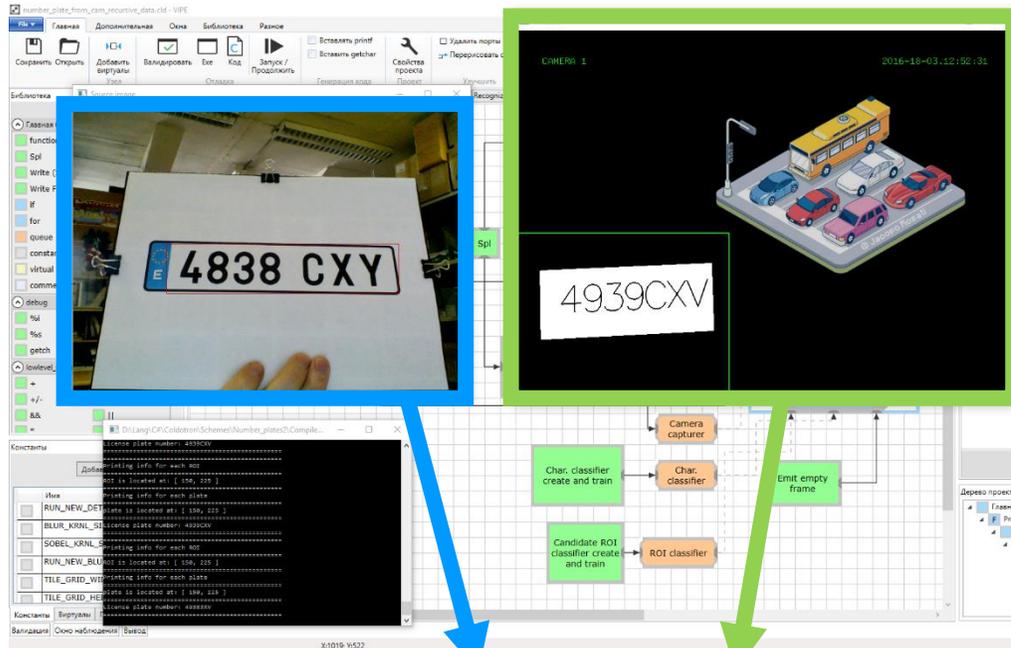
1. требования к производительности ядер встроенной системы
2. требования к памяти ядер встроенной системы
3. занятость вычислительных ядер для различных вариантов размещения и баланс занятости
4. количество и интенсивность обменов данными
5. эффективность загрузки аппаратного обеспечения
6. узкие места аппаратной платформы, программы и распределения задач



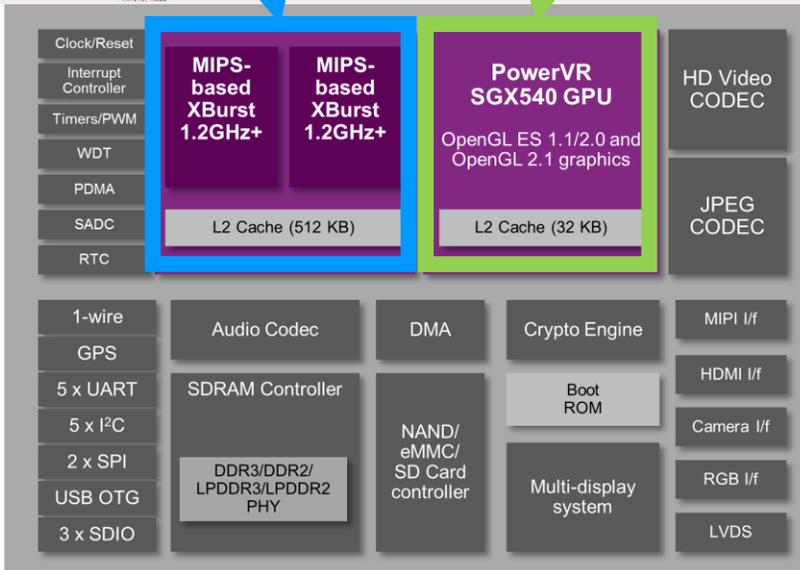
VPL-симулятор

VPL программа

Поддержка неоднородных платформ в VIPE



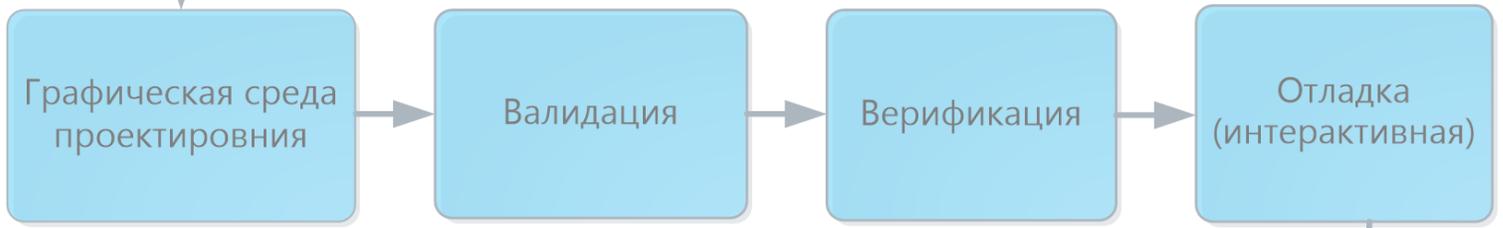
- Назначение элементов схемы на компоненты аппаратной платформы (CPU, GPU, DSP, DMA)
 - операторов на различные типы ядер
 - данных на разные типы памяти
 - обменов на разные типы соединений
- Выбор подходящей реализации операторов обработки данных
- Подготовка исходных данных и отвод результатов работы оператора программы с учетом специфики различных механизмов обмена данными





Специалист в предметной области

Размещение на целевой платформе



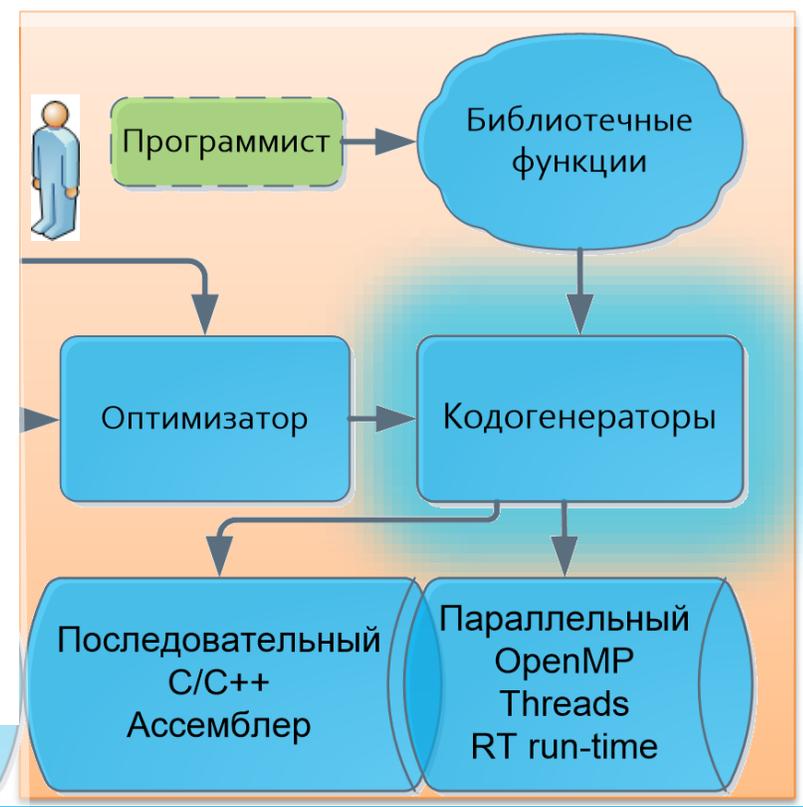
Рабочие прототипы

- ANSI C
- C++
- Параллельный OpenMP
- RT-run-time под платформу *(в работе)*

Proof of concept

- Параллельные потоки
- MPI
- Ассемблер MIPS, DSP

икация



Статистика

Статистика

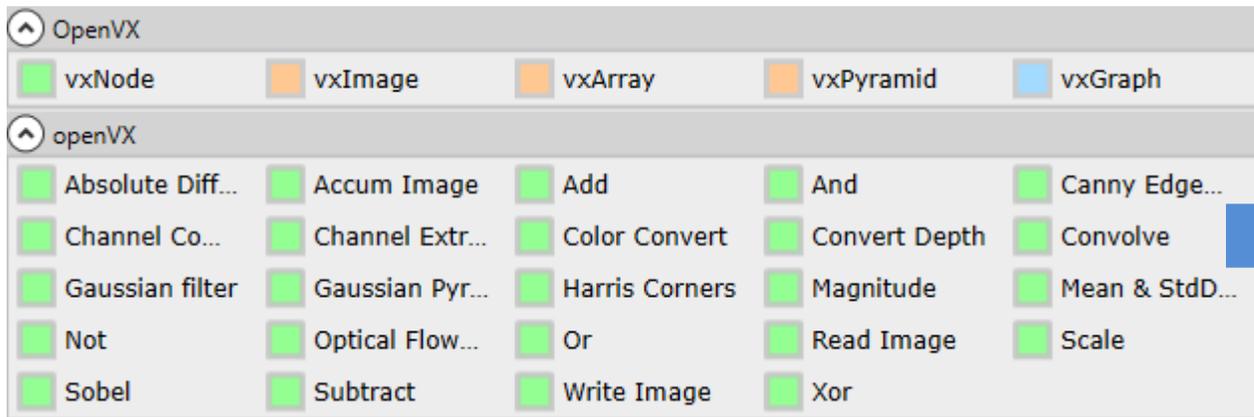
Выводы

- Технология покрывает различные требования к разработке встроенного ПО
- DSL позволяют вовлекать экспертов непосредственно в процесс разработки
- Инструменты оценки производительности помогает разработчику достичь лучших программных характеристик, соблюсти ограничения и требования
- Базовая формальная модель обеспечивает тождественность результатов исполнения программы в любом окружении, в том числе в режиме отладки
- Подход позволяет создавать и отлаживать алгоритм однократно, чтобы потом повторно использовать его в другом окружении и на других платформах
- Технология позволяет быстро подключать новые платформы
- Технология позволяет быстро создавать прототипы для демонстрации их потенциальным клиентам

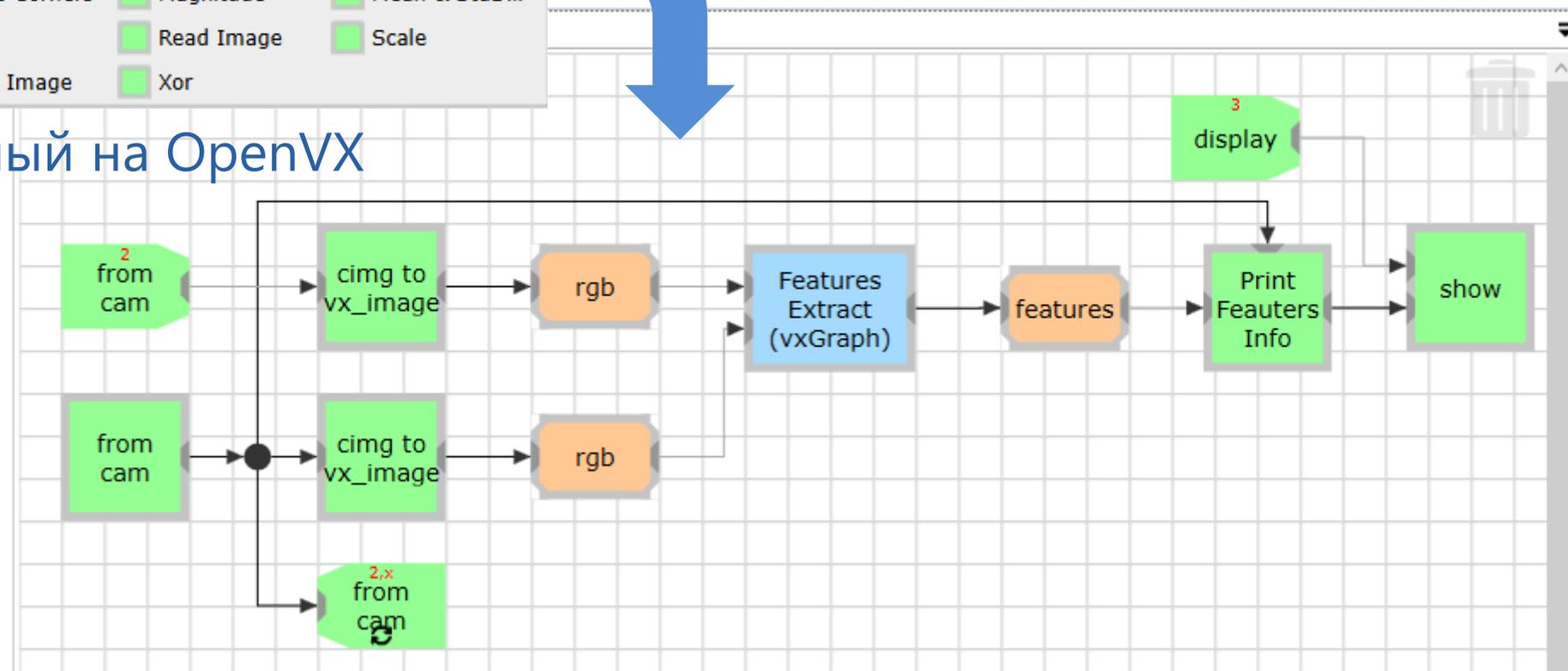
boris.sedov@guap.ru, sheynin@aanet.ru

Примеры применения

Слежение за особенностями сцены DSL и проектирование



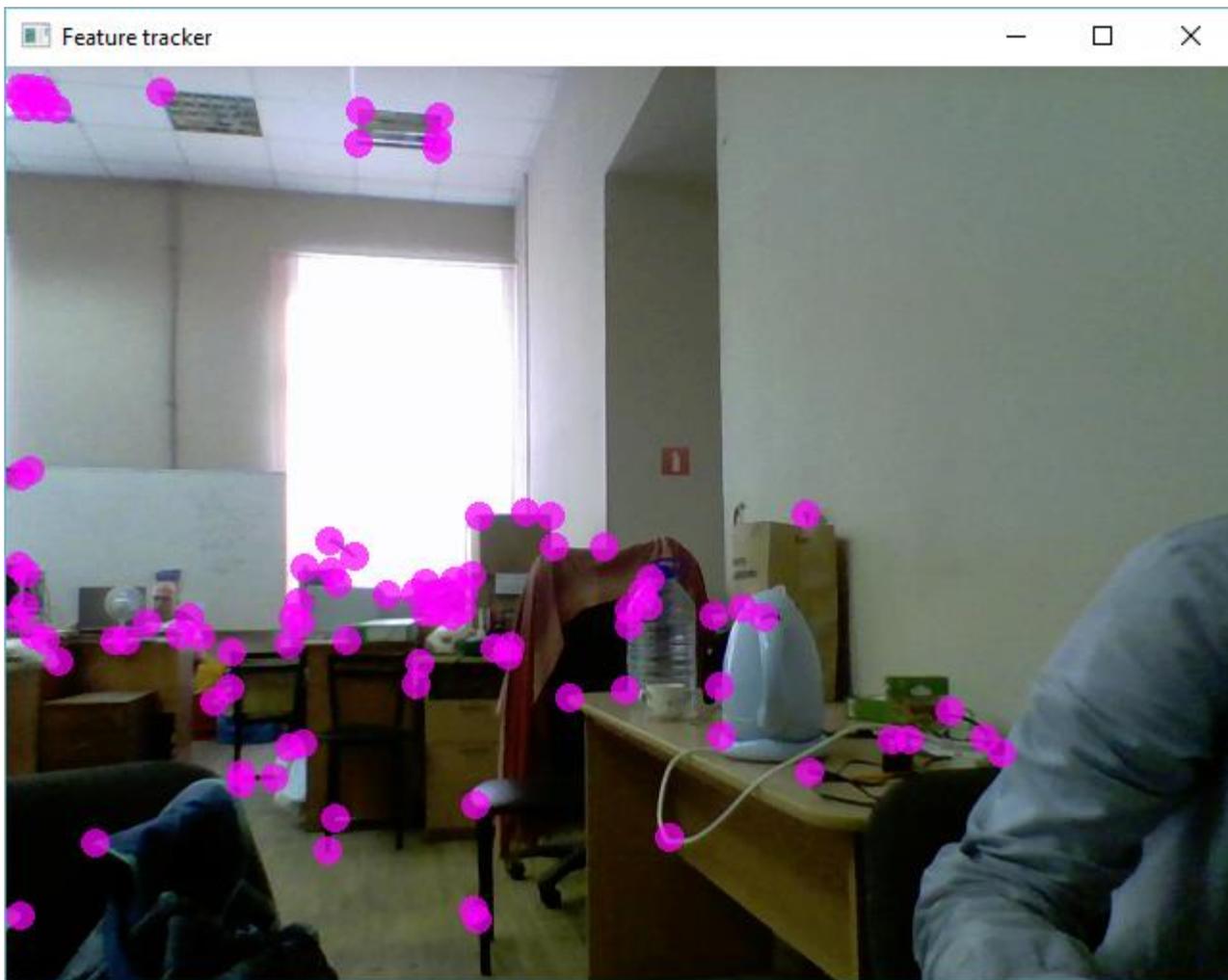
DSL, основанный на OpenVX



Программа отслеживания особенностей сцены в VIPE

Слежение за особенностями сцены

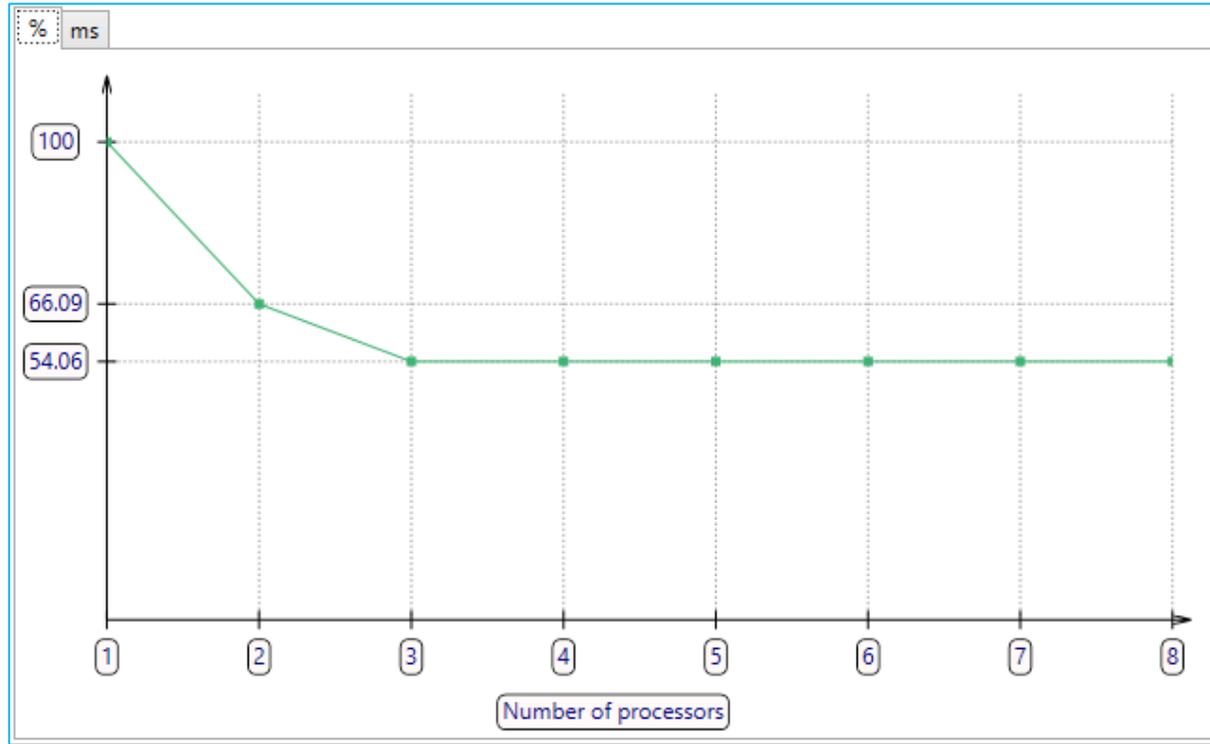
Результаты выполнения



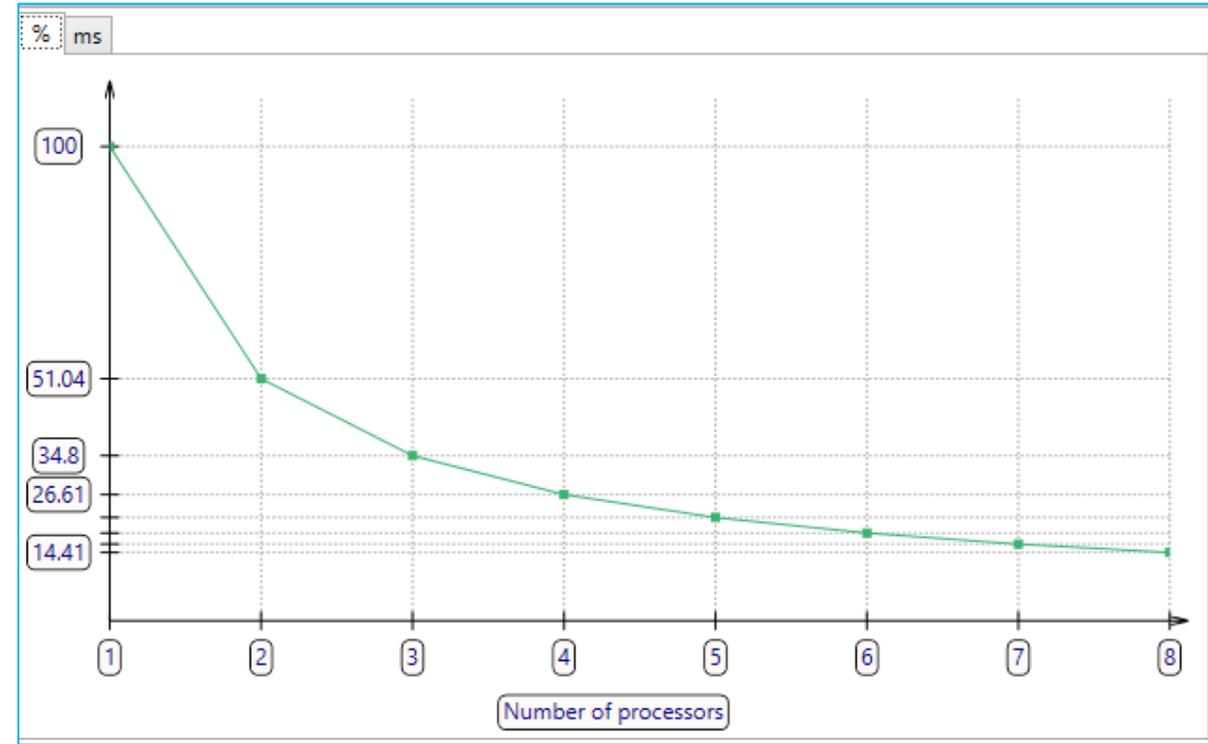
Программа отслеживания особенностей схемы запускалась на платформе x86 с использованием реализации от Khronos

Слежение за особенностями сцены

Статический анализ

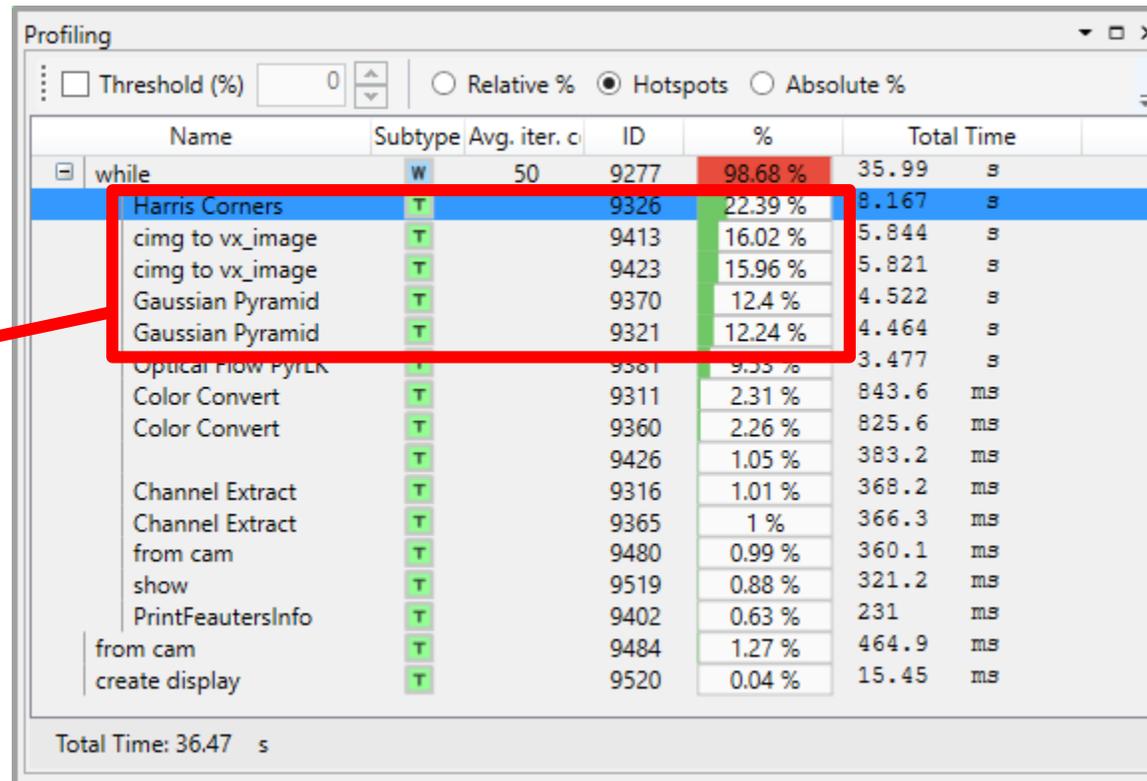


Оценка производительности отслеживания особенностей сцены с **последовательной** обработкой кадров



Оценка производительности отслеживания особенностей сцены с **параллельной** обработкой кадров

Слежение за особенностями сцены Визуальный профилировщик



Profiling

Threshold (%) 0 Relative % Hotspots Absolute %

Name	Subtype	Avg. iter. c	ID	%	Total Time	
while	w	50	9277	98.68 %	35.99	s
Harris Corners	T		9326	22.39 %	8.167	s
cimg to vx_image	T		9413	16.02 %	5.844	s
cimg to vx_image	T		9423	15.96 %	5.821	s
Gaussian Pyramid	T		9370	12.4 %	4.522	s
Gaussian Pyramid	T		9321	12.24 %	4.464	s
Optical Flow PyLK	T		9381	9.55 %	3.477	s
Color Convert	T		9311	2.31 %	843.6	ms
Color Convert	T		9360	2.26 %	825.6	ms
Channel Extract	T		9426	1.05 %	383.2	ms
Channel Extract	T		9316	1.01 %	368.2	ms
from cam	T		9365	1 %	366.3	ms
show	T		9480	0.99 %	360.1	ms
PrintFeautersInfo	T		9519	0.88 %	321.2	ms
from cam	T		9402	0.63 %	231	ms
create display	T		9484	1.27 %	464.9	ms
create display	T		9520	0.04 %	15.45	ms

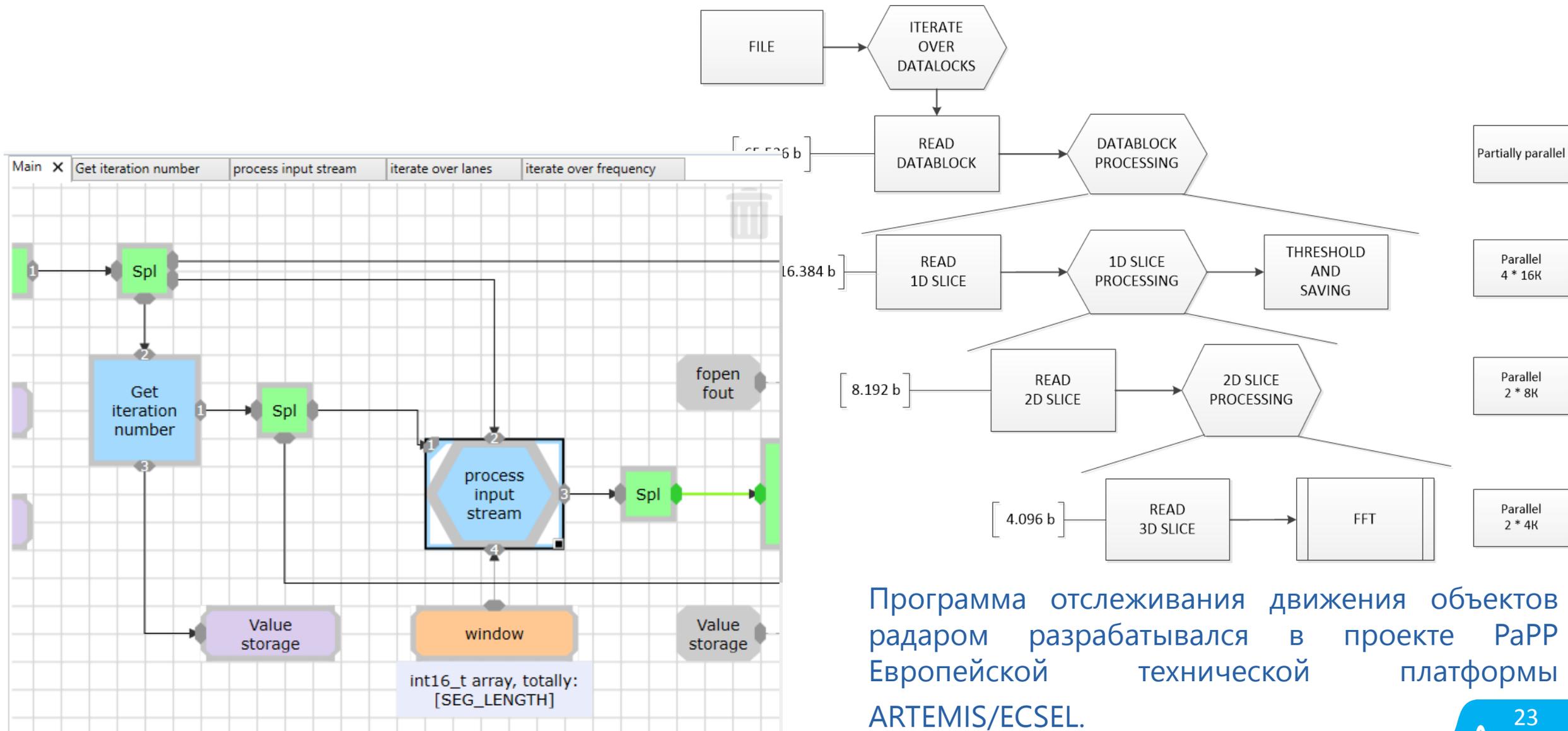
Total Time: 36.47 s

Большое количество времени проходит внутри функций конвертирования изображений (из формата OpenVX и обратно)

Профилирование программы
отслеживания особенностей сцены

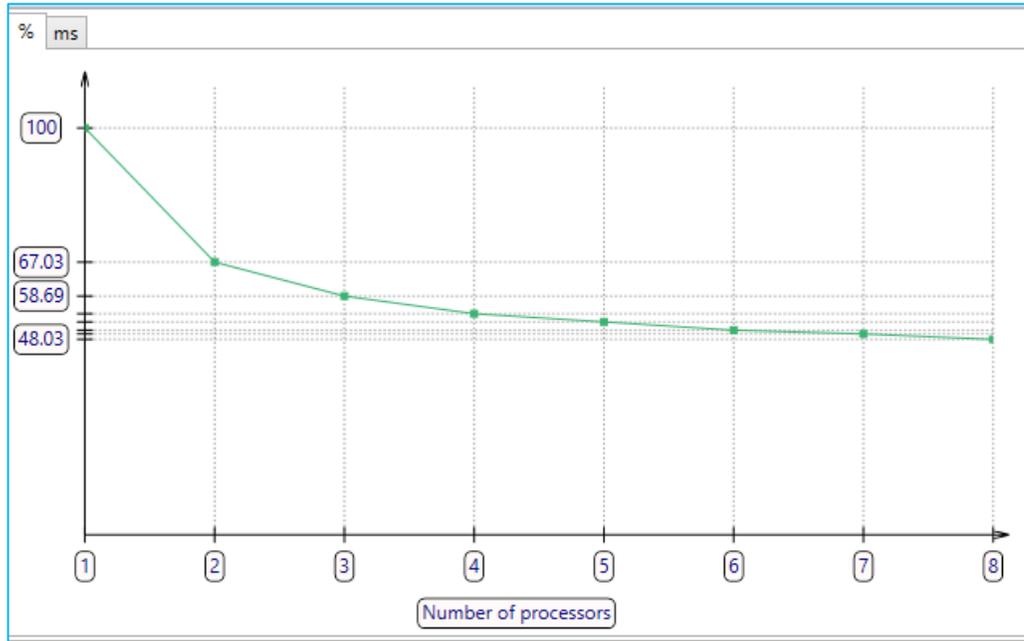
Отслеживание движения объектов радаром

Проектирование



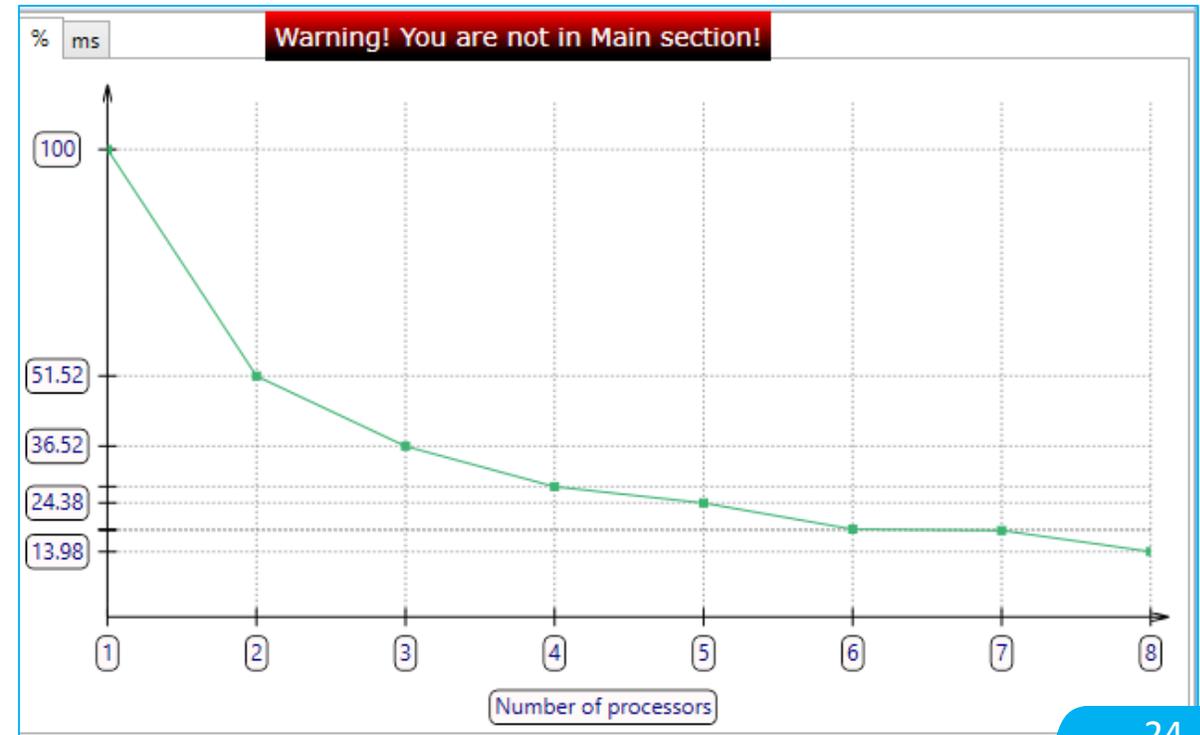
Отслеживание движения объектов радаром

Статический анализ



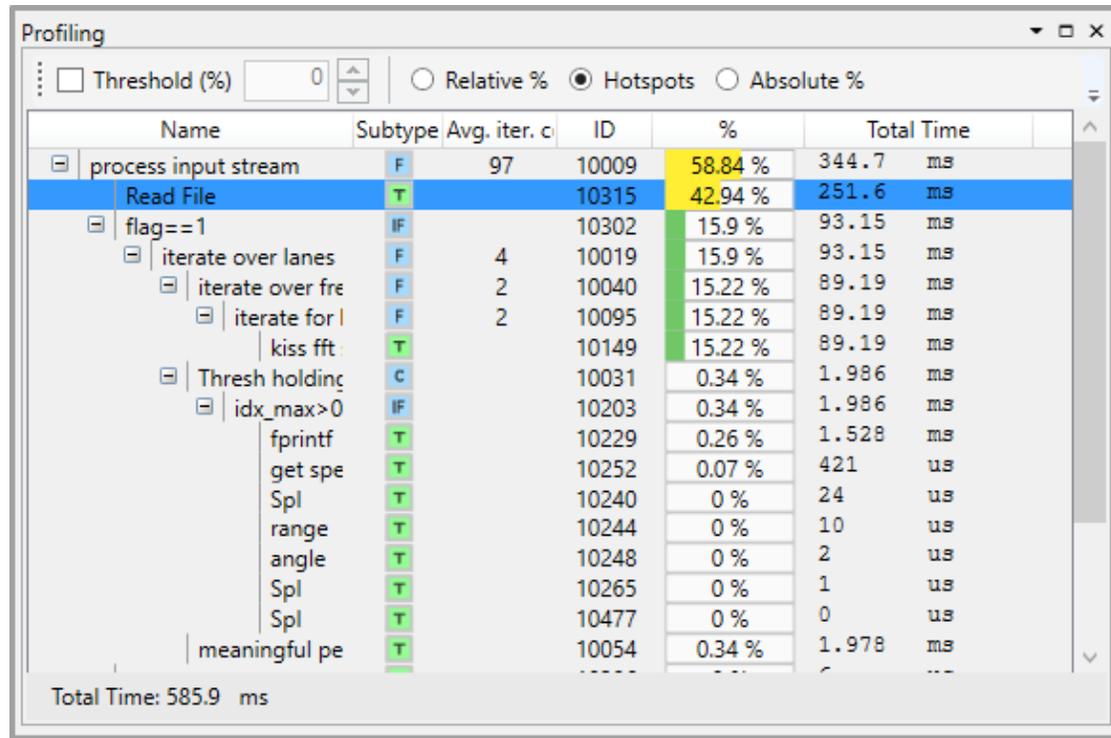
Статический анализ показывает приемлемое сокращение времени на 2-3 ядрах

Однако, результаты оказались хуже ожидаемых. Статический анализ подпрограммы "Обработка блоков данных" показывает близкое к линейному сокращению времени на 8 ядрах



Отслеживание движения объектов радаром

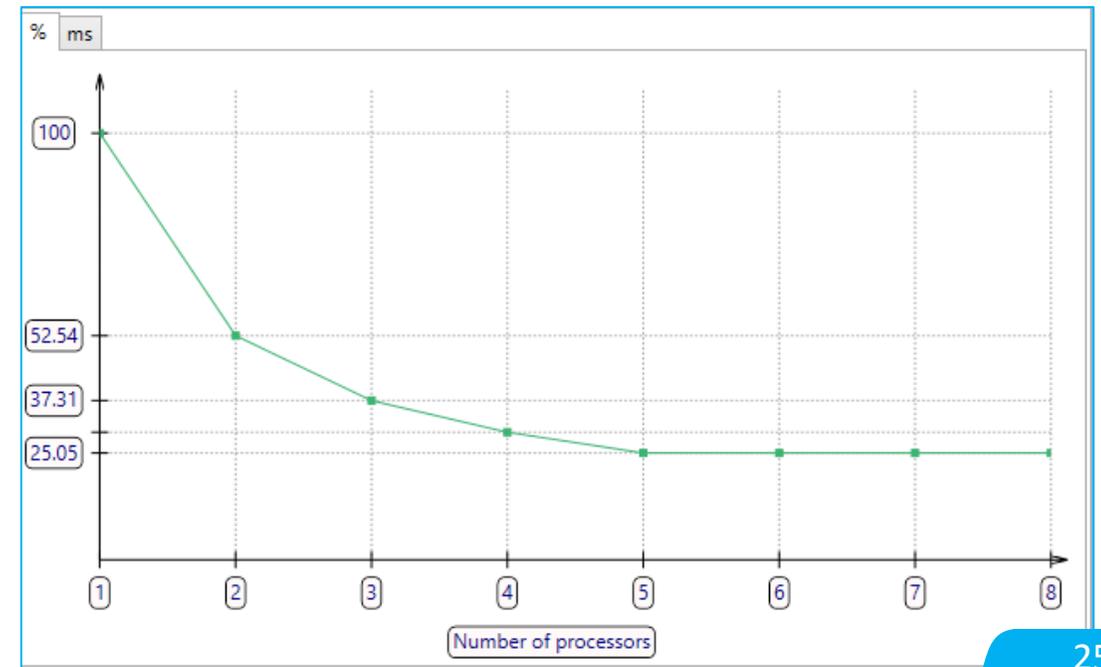
Визуальный профилировщик



Визуальный профилировщик выявляет, что большое количество времени тратится на функцию, которая считывает образцы данных из файла (нынешний прототип использует образцы, хранящиеся в файле).

Функция чтения файла находится в последовательной части, следовательно параллелизм ограничен законом Амдала.

Реальное получение данных должно быть оптимизировано, чтобы занимать меньше времени. Оценка с уменьшенным временем работы функции чтения показывает удовлетворительные результаты.



Отслеживание движения объектов радаром

Сравнение результатов анализа, моделирования и исполнения

Ядра (симулятор) или потоки (OpenMP)	Статический анализ сек. / %	Моделирование VPL сек. / %	Исполнение сек. / %
Без OpenMP			1.60
1	1.26 / 100	1.29 / 100	1.65 / 100
2	0.64 / 50.8	0.88 / 68.2	1.00 / 60.6
3	0.60 / 47.6	0.72 / 55.8	0.81 / 49.1
4	0.34 / 30.0	0.55 / 42.6	1.25 / 76.7

Платформа

- Core i7 8 ядер -> VirtualBox VM 4 ядра
- Ubuntu 14.04, GCC 4.9.2.

Входные данные

- 12 МВ данных сэмплирования сигнала