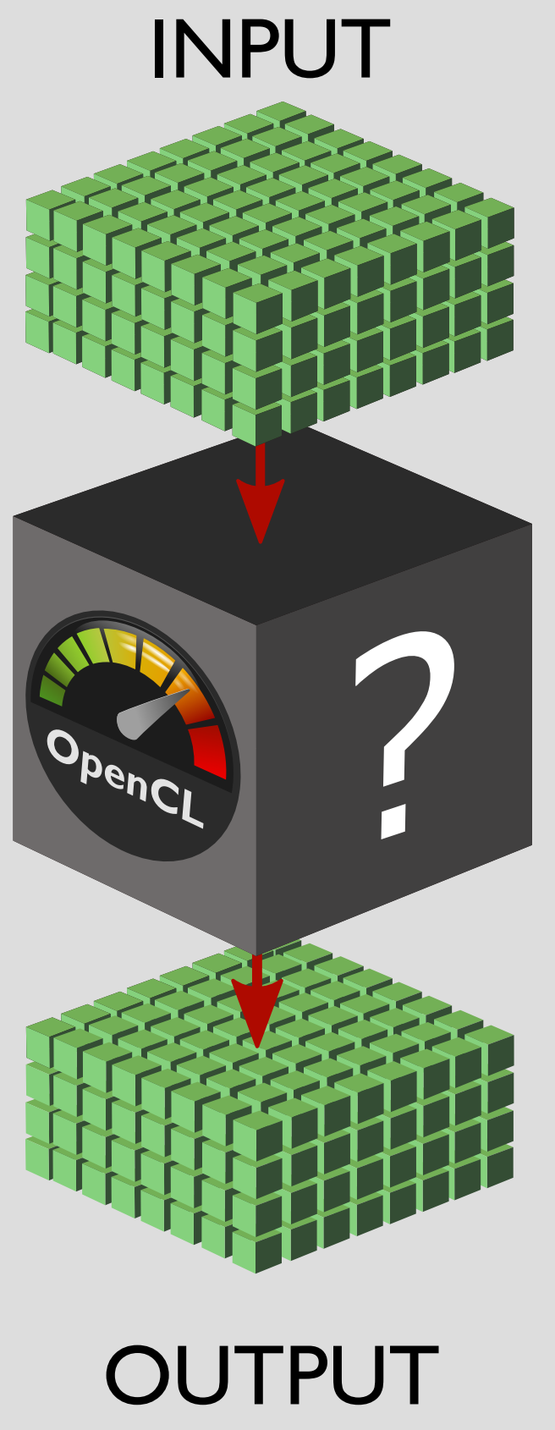
Towards Visual Exploration of Parallel Programs using a Domain-specific Language

Tobias Klein[1,2] Stefan Bruckner[3] Eduard Gröller[1] Markus Hadwiger[2] Peter Rautek[2]

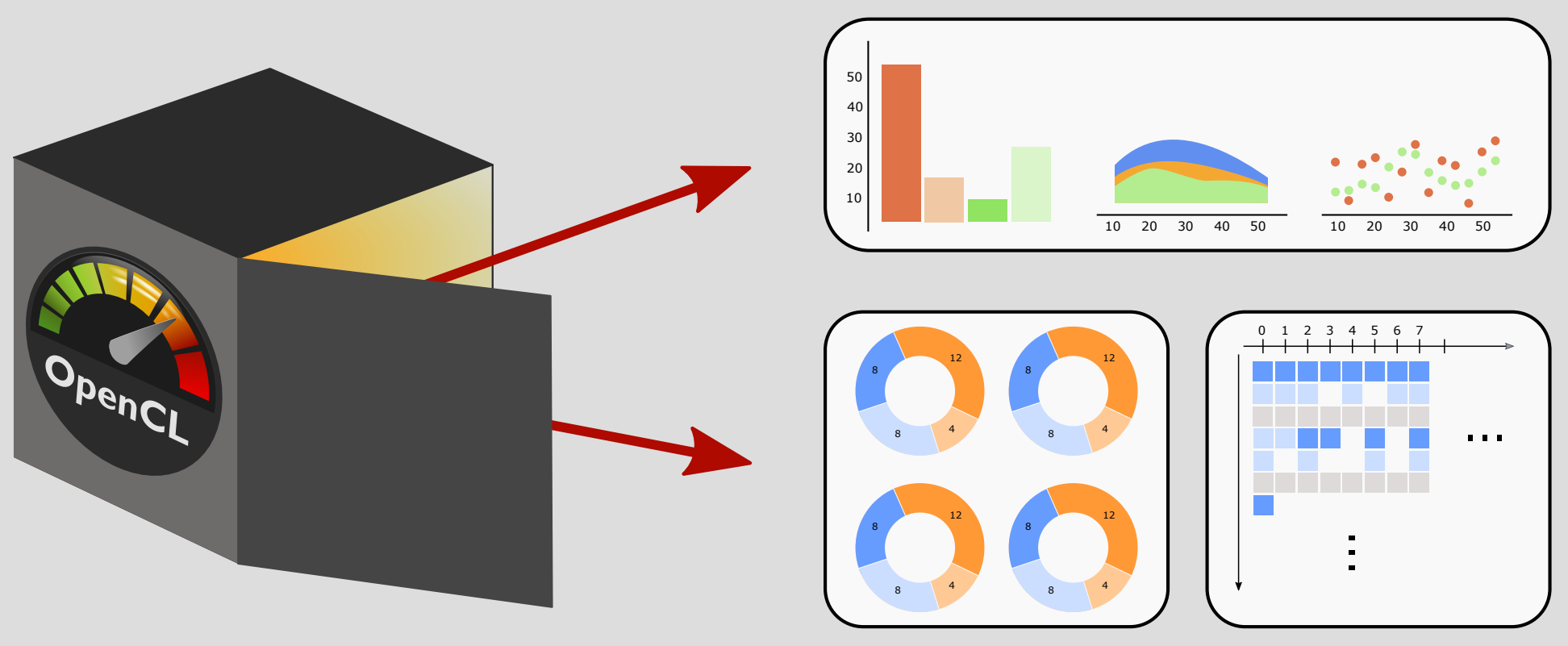
1) Vienna University of Technology 2) King Abdullah University of Science & Technology 3) University of Bergen

MOTIVATION & APPROACH



The execution of parallel programs is often considered as a **black box**, where only input and output is known, but only little is displayed about the execution of the code itself.

However, knowledge about the execution behavior is crucial for the understanding, correctness and especially for the performance of the implementation.



Traditional analysis tools usually read back hardware counters and display statistics that correspond to the question "*What is going wrong*". In contrast to these tools, the aim of this work [1] is to utilize **visualizations** that allow programmers to quickly test their hypotheses on "*Why is something going wrong*".

DOMAIN SPECFIC LANGUAGE

Domain-specific Language (DSL)

- We have developed a DSL, based on ViSlang [2], to rapidly develop parallel programs and to analyze their execution behavior and interaction with the underlying hardware through the aid of visualizations

Code Annotations

- Our DSL provides code annotations that define the recording of the execution behavior, as well as the recording of arbitrary intermediate data

Source-to-Source Compiler

- A compiler transforms programs, written in our DSL, to OpenCL programs with additional instrumentations (corresponding to the annotations) that enable the recording of trace information

Application

- Our DSL approach facilitates rapid prototyping, debugging and profiling of parallel programs

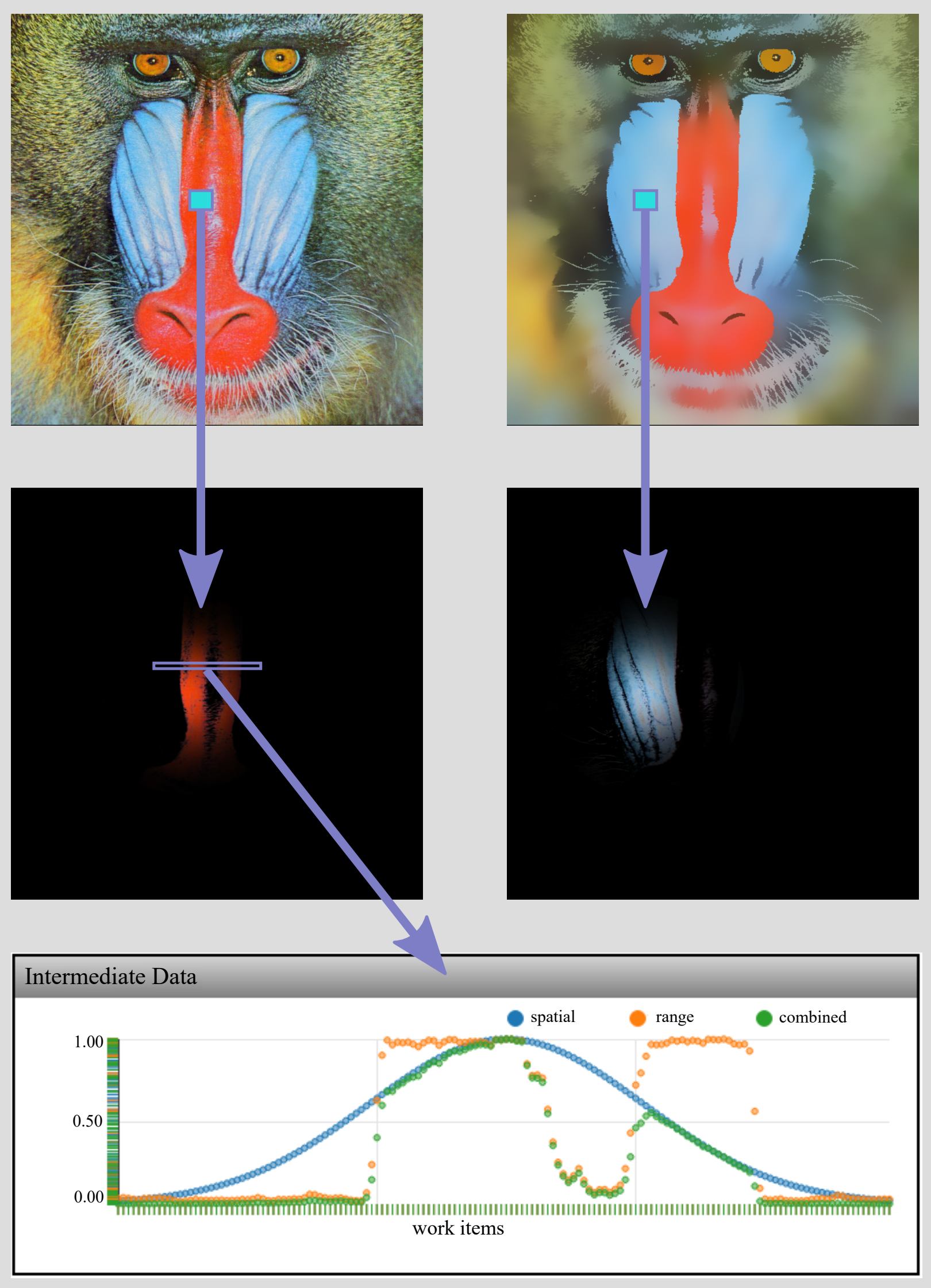
INTERMEDIATE DATA VISUALIZATION

Intermediate Data Extraction

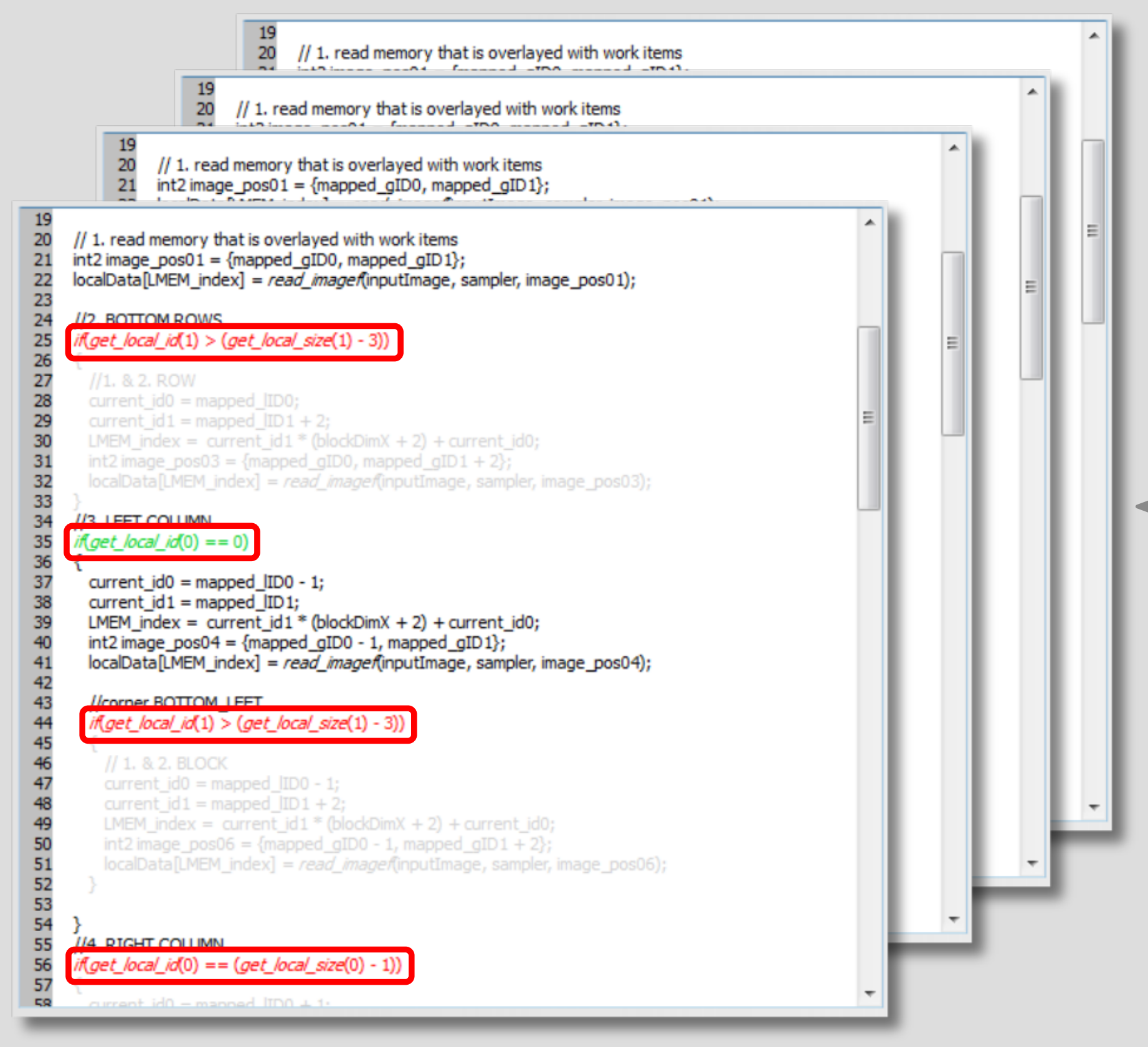
- This example demonstrates the usage of code annotations to extract the intermediate data (in this case: intermediate values of a bilateral filter)
- Instead of manually inserting code to extract the data, code annotations provide a quicker and simpler solution

Data Visualizations

- The two images at the top show input and output
- The images below show the filter values corresponding to a certain pixel
- The graph at the bottom shows a 2D representation of one horizontal snippet of the filter values

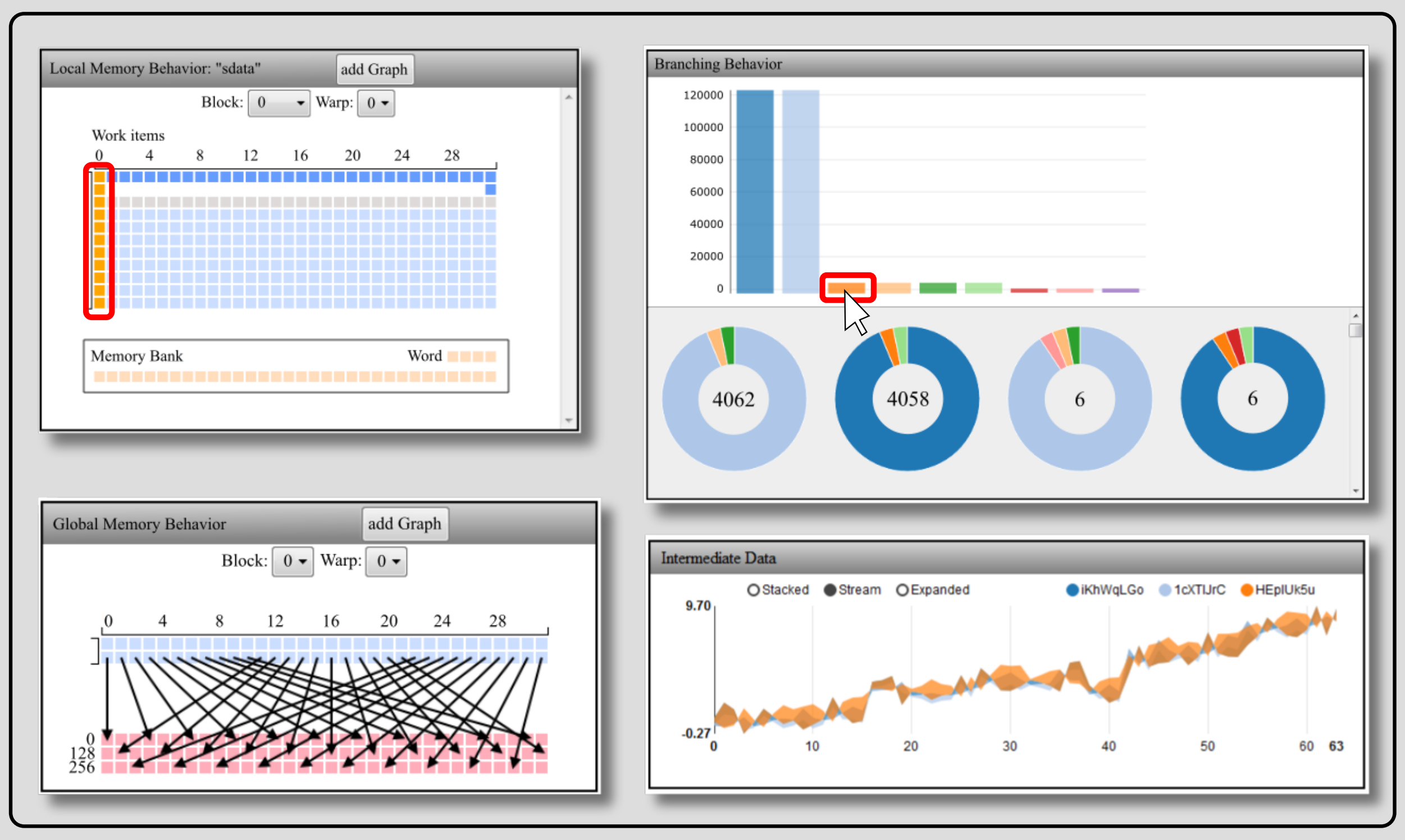


LINKED VIEWS & VISUAL EXPLORATION

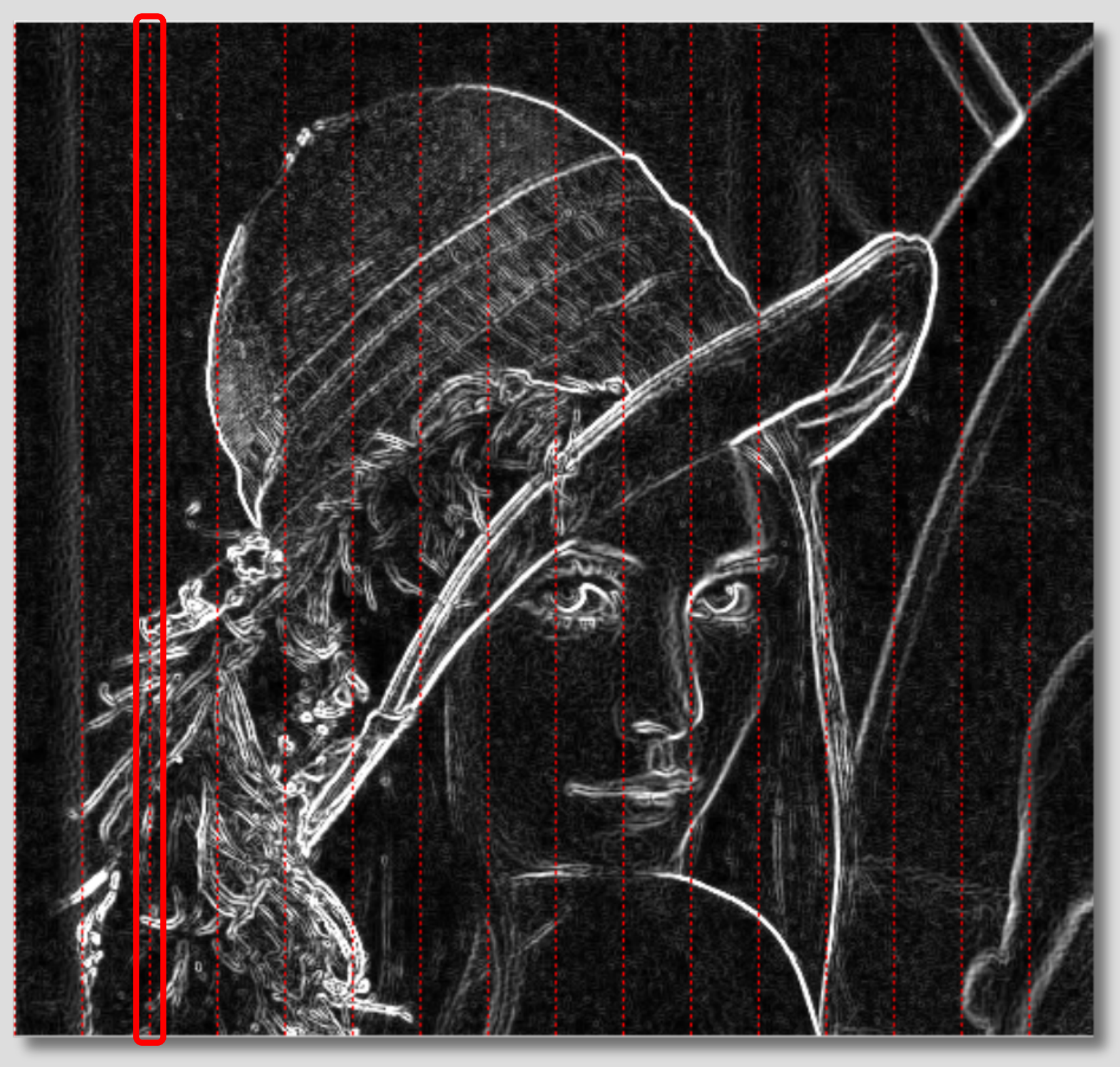


```
// 1. read memory that is overlayed with work items
int2 image_pos01 = {mapped_gID0, mapped_gID1};
localData[LMEM_index] = read_imagef(inputImage, sampler, image_pos01);

//2. BOTTOM ROWS
if(get_local_id(1) > (get_local_size(1) - 3))
{
  //1. & 2. ROW
  current_id0 = mapped_lID0;
  current_id1 = mapped_lID1 + 2;
  LMEM_index = current_id1 * (blockDimX + 2) + current_id0;
  int2 image_pos03 = {mapped_gID0, mapped_gID1 + 2};
  localData[LMEM_index] = read_imagef(inputImage, sampler, image_pos03);
}
//3. LEFT COLUMN
if(get_local_id(0) == 0)
{
  current_id0 = mapped_lID0 - 1;
  current_id1 = mapped_lID1;
  LMEM_index = current_id1 * (blockDimX + 2) + current_id0;
  int2 image_pos04 = {mapped_gID0 - 1, mapped_gID1};
  localData[LMEM_index] = read_imagef(inputImage, sampler, image_pos04);

  //corner BOTTOM_LEFT
  if(get_local_id(1) > (get_local_size(1) - 3))
  {
    // 1. & 2. BLOCK
    current_id0 = mapped_lID0 - 1;
    current_id1 = mapped_lID1 + 2;
    LMEM_index = current_id1 * (blockDimX + 2) + current_id0;
    int2 image_pos06 = {mapped_gID0 - 1, mapped_gID1 + 2};
    localData[LMEM_index] = read_imagef(inputImage, sampler, image_pos06);
  }

}
//4. RIGHT COLUMN
if(get_local_id(0) == (get_local_size(0) - 1))
```

CODE VIEW



DATA VISUALIZATION VIEW



DOMAIN VIEW

Code View : Serves as tool for the development, as well as a visual element, which color encodes and highlights source code corresponding to selected elements
Data Visualization View: Shows visualizations of arbitrary intermediate data, local and global memory accesses and different control flow, using the well-known D3 framework [3].
Domain View : A specialized view that can be used to show the output of a certain domain and that can be linked with the other views.
View Linking: The different views automatically expose their properties within the interface of our tool in order to facilitate their visual linking.

REFERENCES

[1] Tobias Klein. Towards interactive visual exploration of massively parallel programs using a domain-specific language. *Talk at NVIDIA GPU Technology Conference (GTC) , 2016.*

[2] Peter Rautek, Stefan Bruckner, M. Eduard Gröller, and Markus Hadwiger. ViSlang: A system for interpreted domain-specific languages for scientific visualization. *IEEE Transactions on Visualization and Computer Graphics (Proc. SCIVIS '14)*, 20(12):2388-2396, 2014.

[3] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics (Proc. InfoVis '11)*, 17(12):2301-2309, 2011.

CONTACT *tobias.da.klein@gmail.com*
peter.rautek@kaust.edu.sa