OPEN SOURCE HEVC ANALYZER FOR RAPID PROTOTYPING (HARP)

Dominic Springer, Wolfgang Schnurrer, Andreas Weinlich, Andreas Heindel, Jürgen Seiler, and André Kaup

Multimedia Communications and Signal Processing
Friedrich-Alexander-University Erlangen-Nürnberg (FAU), Cauerstr. 7, 91058 Erlangen, Germany

ABSTRACT

The design of new HEVC extensions comes with the need for careful analysis of internal HEVC codec decisions. Several bitstream analyzers have evolved for this purpose and provide a visualization of encoder decisions as seen from a decoder viewpoint. None of the existing solutions is able to provide actual insight into the encoder and its RDO decision process. With one exception, all solutions are closed source and make adaption of their code to specific implementation needs impossible. Overall, development with the HM code base remains a time-consuming task. Here, we present the HEVC Analyzer for Rapid Prototyping (HARP), which directly addresses the above issues and is freely available under *www.lms.lnt.de/HARP*.

1. INTRODUCTION AND MOTIVATION

The new High Efficiency Video Coding (HEVC) standard, also known as H.265, took roughly ten years to evolve from its predecessor, H.264/AVC. With around 70,000 lines of code, its reference implementation HM [1] requires a well-chosen setup for designing, implementing, and debugging new extensions of the HEVC standard. Industry and research community have addressed this with several HEVC analyzers [2, 3, 4, 5], allowing a decoder-side view on bitstream internals. While development work often takes place in the encoder first, none of the existing solutions is able to monitor encoder behavior, e.g. for rate-distortion optimization (RDO) analysis. With the exception of [5], all solutions are closed-source and cannot be adapted to reflect individual needs for specific HM adaptions. Support of Linux, which builds the basis of many research infrastructures, is rare, and export capabilities of HM data structures are minimal or non-existent. As a result, the implementation and debugging of new HEVC extensions remains time-consuming for both new and experienced HM developers.

In this work, we present the HEVC Analyzer for Rapid Prototyping (HARP). Its motivation lies in the simple idea that a well-chosen development environment allows to focus time and resources more on the design stage of HEVC extensions and less on subsequent implementation and debugging tasks. For rapid prototyping on C++ side, HARP makes heavy use the advanced C++ libraries OpenCV [6], Qt [7], and PicklingTools [8]. HARP allows to export HM data structures to Python Dictionaries and thus Matlab-like processing, statistical evaluation and plotting in native Python [9]. HARP is licensed under GNU GPL and offers four essential features:

- Easy setup of an HEVC development environment
- Support of encoder analysis (e.g. RDO decision paths)
- C++ export of HEVC information to Python Dictionaries
- Python frontend for easy numerical processing and plotting

2. THE HARP TOOLKIT

The HARP toolkit consists of the central HARP C++ core and Python GUI components, accompanied by a wide set of different libraries and tools for fast prototyping. On Linux, downloading HARP and calling one command line is sufficient to create a full-fledged, fully open source HM development environment as depicted in Fig. 1. During compilation of HARP, Eclipse project files are automatically generated. Imported into Eclipse, all HARP C++ core code as well as all used C++ and Python libraries are automatically scanned and type-indexed. In its current version 1.0, HARP is able to:

- Access CTU/CU data types (see CShow_UnitsCloseup.h)
- Access PU parts and MVs (see CShow_PredictionUnits.h)
- Monitor RDO decision paths (see CShow_RDO.h)
- Analyze chosen PU refs (see CShow_RefIndices.h)
- Analyze chosen TU sizes (see CShow_TransformUnits.h)

2.1. HARP Examples

Fig. 2 uses four CTUs from sequence Cactus to demonstrate the capabilities of the HARP C++ core with regard to Prediction Units (PUs), PU ref indices, Transform Units (TUs), and HM data structures. We recommend the class CShow_UnitCloseup as a starting point for new HM developers: it makes typical use of encoder/decoder data structures and LUT-based z-scan/raster-scan indexing of atomic 4x4 HM storage units. Another example, showing HARP visualization of the RDO decision tree is shown in Fig. 3a. For the partially rotating content in CTU 343 (see Fig. 2), the first five RDO evaluations are depicted. PU size is first full

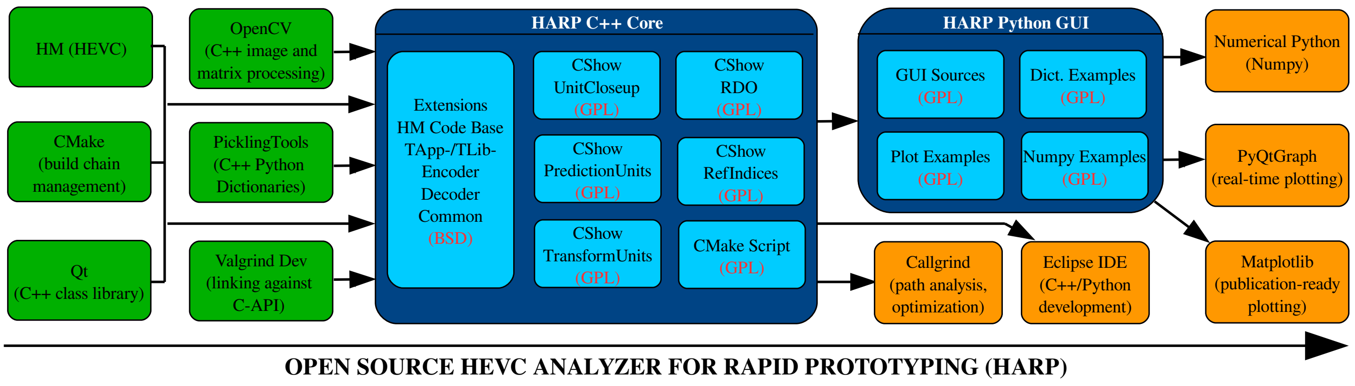

HM (HEVC)
CMake (build chain management)
Qt (C++ class library)
OpenCV (C++ image and matrix processing)
PicklingTools (C++ Python Dictionaries)
Valgrind Dev (linking against C-API)
HARP C++ Core
Extensions HM Code Base TApp-/TLib-Encoder Decoder Common (BSD)
CShow UnitCloseup (GPL)
CShow RDO (GPL)
CShow PredictionUnits (GPL)
CShow RefIndices (GPL)
CShow TransformUnits (GPL)
CMake Script (GPL)
HARP Python GUI
GUI Sources (GPL)
Dict. Examples (GPL)
Plot Examples (GPL)
Numpy Examples (GPL)
Numerical Python (Numpy)
PyQtGraph (real-time plotting)
Matplotlib (publication-ready plotting)
Callgrind (path analysis, optimization)
Eclipse IDE (C++/Python development)
OPEN SOURCE HEVC ANALYZER FOR RAPID PROTOTYPING (HARP)

Fig. 1: Overview of HARP toolkit, with the HARP code base marked in blue color. Green color: C++ libraries and tools integrated into the toolkit. Orange color: optional tools and Python libraries for rapid IDE-based development, code profiling and publication-ready plotting.

2Nx2N @ 64x64, then split to Nx2N. The second and third predInterSearch() tests are able to approximate the rotational motion better than the first 2Nx2N test, but still suffer from strong prediction error and residual energy.

The analyzer can also be run in demonstration mode, taking in the results from a video stream (like a webcam) and showing live visualizations of the encoder behavior during HEVC compression. Fig. 3b shows a screenshot of the HARP Python GUI, running in Live HARP Demo mode.

2.2. Processing HEVC Data in Python

The programming language Python [9] has gained significant popularity for scientific applications. With its rising number of specialized toolboxes, Python provides scientific tools similar to Matlab without requiring the purchase of any licenses. Python code is highly portable between different platforms and runs on small embedded devices as well as on large-scale parallel HPC clusters. We make use of PicklingTools [8] to easily export HEVC information under C++ as Python Dictionaries, which can then be read in and processed efficiently in Python. We recommend the Live HARP Demo (see Fig. 3b) as a ready-to-use basis for interested developers.

2.3. Compiling and Installing HARP

All major Linux distributions provide package managers to setup full-fledged development environments and library installations with a single command line. For HARP 1.0 on Ubuntu 14.04 LTS, the following line sets up all prerequisites for an HEVC development environment identical to Fig. 1:

```
sudo apt-get install build-essential libopencv-dev qt4-default cmake valgrind libpython2.7-dev python-numpy python-scipy python-matplotlib python-pyinotify
```

For full support of HARP Python GUI, we recommend a manual installation of Eclipse CDT 4.2 (or higher) and PyQtGraph. We based HARP on CMake [10], a multiplatform C/C++ build management, so that HARP can be easily compiled and extended with new libraries. The HARP components were carefully chosen for platform independence, which opens up possible ports of HARP to Mac OS or Windows OS. Please note that we did not test HARP on Windows since CMake, OpenCV, Qt, Python, PyQtGraph, and Matplotlib are required to be installed by hand with appropriate PATH configuration (testing, documentation, and screenshots of these steps on Windows are very welcome).

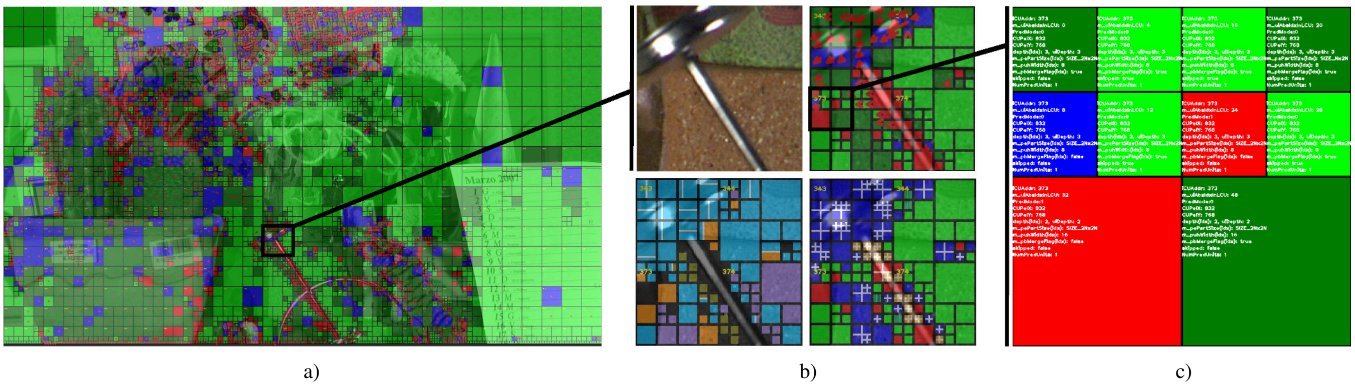
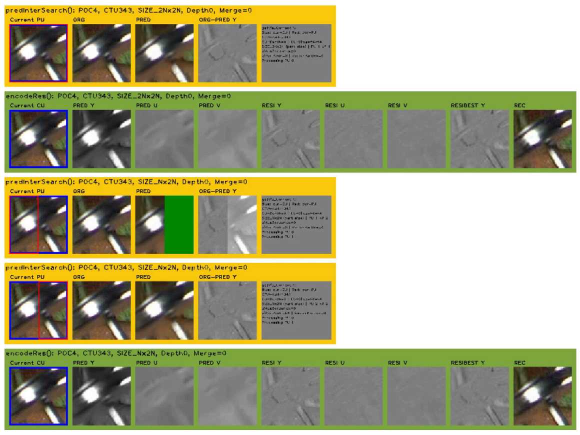


Fig. 2: HARP visualization during HM-14.0 encoding of POC4 in sequence Cactus. a) Output of C++ class CShow_PredictionUnits (red: intra, blue: inter, green: merge, light green: skip, red arrows: MVs). b) Left to right, top to bottom: original image patch, magnified output of classes CShow_PredictionUnits, CShow_RefIndices and CShow_TransformUnits. c) Magnified output of class CShow_UnitCloseup.


a)

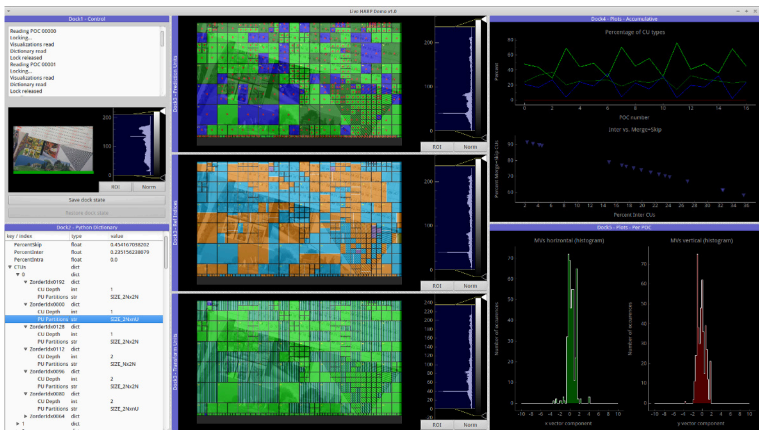

b)

Fig. 3: a) HARP visualization example of the first two RDO tests performed on CTU 343 (export of all tests possible). After HM's predInterSearch() found the best PU MV (yellow), encodeResAndCalcRdInterCU() calculates the corresponding residual (green). b) Live HARP Demo for ICIP 2014 Show&Tell, graphs demonstrate how motion vectors and mode info can be numerically processed and plotted in Python.

3. CONTRIBUTIONS ARE WELCOME!

The architectural choices in Fig. 1 reflect our need for fast access to the HM codebase: rapid prototyping is essential for early stages of scientific research. If you plan to modify the HEVC encoder and decoder code bases, the HARP toolkit is probably the right choice. For a first start with HEVC without any knowledge about the standard, we recommend the open source Gitl HEVC bitstream analyzer [5]. We put all sources under GPL license (with the exception of changes to HM core libraries, which remain BSD) in order to create an incentive for interested developers to use and extend the HARP code base. Our HEVC research is ongoing, so please contribute back interesting new code sections! Among a list of possible contributions, these three ideas stand out:

- C-API for HARP to make its C++ code linkable from C
- Usage of this C-API for HARP integration into VP9 [11]
- MinGW build and documentation of HARP for Windows

4. CONCLUSION

We presented HARP, a GNU GPL licensed HEVC analyzer toolkit for rapid prototyping (*www.lms.lnt.de/HARP*). While existing HEVC analyzers are restricted to decoder side and, with one exception [5], are closed source only, HARP allows detailed analysis like encoder-side RDO behavior or decoder-side CU mode visualization and is freely adaptable to specific needs. HARP's main focus is to assist future research on new HEVC extensions by offering a complete development environment, consisting of useful libraries, essential build and visualization tools, Python toolboxes, and ready-to-use Eclipse IDE project files. We encourage interested developers to freely use and extend our toolkit and contribute interesting new code sections back to the HARP code base.

5. REFERENCES

[1] ITU/ISO/IEC, "HEVC Test Model HM-14.0," www.hevc.hhi.fraunhofer.de.

[2] Codecian, "CodecVisa: HEVC and VP9 bitstream analyzer," www.codecian.com.

[3] Solveigmm, "Zond 265: HEVC bitstream analyzer," http://www.solveigmm.com/en/products/zond.

[4] Elecard, "Elecard HEVC bitstream analyzer," www.elecard.com.

[5] Sun Yat-sen University, "Gitl HEVC bitstream analyzer," www.github.com/lheric/GitlHEVCAnalyzer.

[6] OpenCV Dev Team, "OpenCV: image processing library," www.opencv.org.

[7] Qt Dev Team, "Qt4 library," www.qt-project.org.

[8] R. Saunders, "Complex software systems in legacy and modern environments: A case study of the picklingtools library," in *Proc. HICSS*, Kauai, Hawaii, USA, 2010, pp. 1–10, source available under www.picklingtools.com.

[9] Python Software Foundation, "Python programming language," www.python.org.

[10] CMake Dev Team, "CMake: cross-platform, open-source build system," www.cmake.org.

[11] D. Mukherjee, J. Bankoski, A Grange, H. Jingning, J. Koleszar, P. Wilkins, X. Yaowu, and R. Bultje, "The latest open-source video codec VP9 - an overview and preliminary results," in *Proc. PCS*, San Jose, USA, Dec 2013, pp. 390–393.