WWW.SILEXICA.COM

WHITEPAPER

HEVC CODEC ANALYSIS

SILEXICA

1 Introduction

This document presents the results of running the SLX Parallelizer, an integrated part of the SLX Tool Suite, to quickly explore the HHI/Frauenhofer reference HEVC codec implementation. While the codec is designed to allow developers to make trade-offs of coding efficiency vs. effective use of the parallel computation resources of the target platform, the SLX Parallelizer is primarily used to assess the benefits of any exposed parallelism in the current implementation. The presented results are based on the unedited implementation of the code as downloaded. For the purposes of this experiment, a virtual hexacore target platform based on ARM Cortex A7 processors was chosen.

1.1 SLX Parallelizer

Figure 1 shows the main components of SLX Parallelizer methodology. The inputs are a sequential C application and a model of the target multicore platform. This model describes relevant details of the platform such as the number of cores, execution costs of the instruction set, the memory architecture, and communication costs. Then a program representation is constructed, which combines the outcomes of static and dynamic analyses. While the static analysis is based on compile-time information such as the complete control flow, the dynamic analysis is based on a typical execution and runtime information such as a list of executed functions, basic block execution counts and memory accesses involving pointers. The dynamic information is obtained by instrumenting the intermediate generated by the compiler and executing the resulting binary to generate a trace. The program representation is analyzed by algorithms that extract different forms of parallelism. The tool presents its finding in several main categories:

- Target cost aware examination of the program sources through annotated source-code editor and a graphical call-graph of the application
- Parallelization hints highlighting both important inhibitors of parallelism and detected target-appropriate parallelism patterns including their local and global speedup opportunities
- General speedup estimates

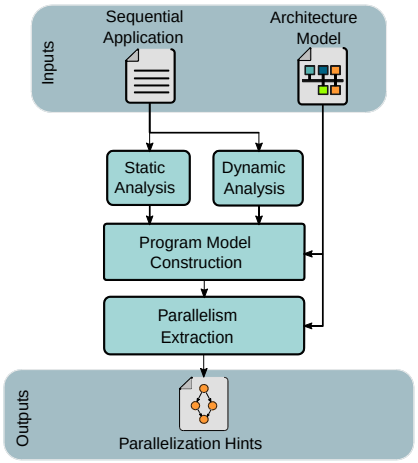

Figure 1: Parallelization methodology

1.2 Speedup calculations

In the SLX Parallelizer, the number of processor cores is taken from the architecture description file. When a parallelism pattern is identified, it is considered in isolation for its suitability to be implemented on the given architecture with the following considerations:

The tool looks at data-level (DLP), pipeline-level (PLP) and task-level parallelism (TLP) and applies an internal heuristic to map the identified partitions on the number of available cores.
Cost heuristics include predicted partition execution cost, estimated inwards and outwards communication costs, and synchronization/control costs. Parallelization is only recommended if the highest partition cost is lower that the predicted sequential execution cost of the whole pattern, identified as a "local speedup ratio". Amdahl's Law is taken into consideration in computing the global speedup of applying this pattern in isolation.

2 HEVC Case Study

2.1 Overview

High Efficiency Video Coding (HEVC), also known as H.265, is a video compression standard, one of several potential successors to the widely used AVC (H.264 or MPEG-4 Part 10). In comparison to AVC, HEVC offers about double the data compression ratio at the same level of video quality, or substantially improved video quality at the same bit rate. It supports resolutions up to 8192×4320, including 8K UHD.

The source code of HEVC used in this case study contains 3268 function definitions (including function template instantiations etc.) in more than 86KLOC. Focusing on those that use >5% target execution time, 35 functions were analyzed in detail. No appropriate TLP was identified. For loop-oriented parallelism, 89 loops spread over 19 functions were analyzed.

For 6 loops, potentially useful parallelism was recognized - if all were implemented a speedup of 1.38 would be expected. Inhibiting factors were reported for other loops (note: more than one reason is listed for some loops):

- In 32 cases we reported loop carried dependencies as inhibiting factors.
- 22 rejected due to irregular control flow
- 20 because they body was too small/unimportant
- 10 because of nesting issues
- 10 because of induction variable issues

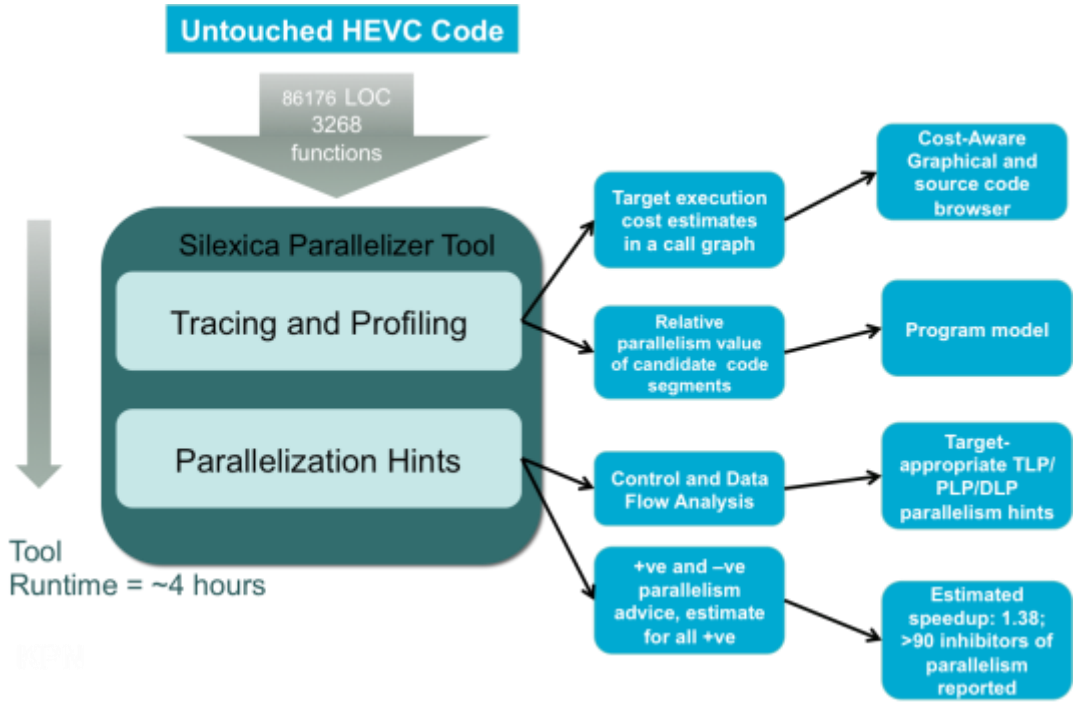

Figure 2: HEVC Codec Case Study Results

2.2 Exposing additional parallelism

The SLX Parallelizer contributes to change the application source code to expose more parallelism. It is able to reason about inhibiting factors and theoretical speedup numbers assuming a proper rewrite of the source code. For each of the inhibiting factors mentioned above, an example output message of the tool and its explanation is given:

Rejected due to loop carried dependencies:

When loop carried dependencies are detected, implementation patterns like reductions are first considered. If no known implementation pattern is detected, the objects are listed as loop carried dependencies. If one end of the dependence is clearly in a callee then the callee is also mentioned.

source/Lib/TLibEncoder/TEncCu.cpp:1238: DLP - Loop (1238:1284) does not provide DLP because of loop-carried dependency on variables bestIsSkip [RAW], pcSubBestPartCU [WAR] in function xCheckBestMode, pcSubTempPartCU [RAW] in function xCheckBestMode. Total execution percentage: 8.06%

source/Lib/TLibEncoder/TEncCu.cpp:1238: DLP - Loop (1238:1284) would use 5 workers, if dependencies could be ignored. Estimated speedup would be: Local 4.98. Global 1.07

Rejected due to irregular control flow:

source/Lib/TLibEncoder/TEncSlice.cpp:752: Loop (752:928) has multiple exits, e.g., due to goto, break or return identified at line 884
source/Lib/TLibEncoder/TEncSlice.cpp:752: DLP - Loop (752:928) would use 9 workers, if dependencies could be ignored. Estimated speedup would be: Local 9.00. Global 8.57

```
752    for (UInt ctuTsAddr = startCtuTsAddr; ctuTsAddr < boundingCtuTsAddr; ++ctuTsAddr)
753    {
...
882        if (boundingCtuTsAddr <= ctuTsAddr)
883        {
884            break;
885        }
…
928    }
```

Their body was too small/unimportant

This reports that the ratio of extra communication cost to parallel execution is too high.

source/Lib/TLibCommon/TComRdCost.cpp:1611: DLP - Loop (1611:1614) presents no beneficial parallel-for due to communication overhead
source/Lib/TLibCommon/TComRdCost.cpp:1611: PLP - Loop (1611:1614) can not be parallelized as a pipeline because its body is too small

```
1611            for (x = 0; x < iCols; x += 4)
1612            {
1613                uiSum += xCalcHADs4x4(&piOrg[x], &piCur[x * iStep], iStrideOrg, iStrideCur, iStep);
1614            }
```

Nesting issues

If PLP is selected for an outer loop, inner loops are not further investigated. (Note: This is a restriction we lift in the upcoming release)

```
source/Lib/TLibCommon/TComTrQuant.cpp:2261: PLP parallelism finally selected for loop (2261:2388) with a local speedup of 1.12, global speedup of 1.01, using 2 stages (distributed as: 2263:2295 - 2297:2388)

source/Lib/TLibCommon/TComTrQuant.cpp:2438: Loop (2438:2449) is not considered for DLP as it is nested in loop (2261:2388) which is a PLP candidate
```

Induction variable issues

Loop carried dependencies that match an induction variable pattern are reported separately so the developer can re-express them in terms of the loop control variable.

```
source/Lib/TLibCommon/TComInterpolationFilter.cpp:224: DLP - Loop (224:260) can not be parallelized as it has multiple induction variables: dst row src. Induction variable substitution could enable parallelization. Total execution percentage: 8.94%

source/Lib/TLibEncoder/TEncGOP.cpp:1597: PLP - Loop (1597:1632) can not be parallelized as a pipeline because the induction variable is updated inside its body
```

3 Conclusion

In summary, in about 4 hours of computer run time, you are able to see that the maximum speedup you can hope to achieve with the default implementation is about 1.38. For additional 32 loops, opportunities for further parallelism with its associated potential speedup on the target platform were identified including the inhibiting reason. Overall, the tool quickly identifies the places where manual efforts should be spent and which ones should be avoided. This case study on unchanged, real-life application source code shows the significant productivity increase by using automation versus a costly, manual investigation by parallelism experts.