



**QUEEN'S
UNIVERSITY
BELFAST**

Parallelization and Performance of an H.264 Video Encoder on the Cell B.E

Alvanos, M., Tzenakis, G., Nikolopoulos, D., & Bilas, A. (2009). Parallelization and Performance of an H.264 Video Encoder on the Cell B.E. In *Proceedings of the Fifth International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems (ACACES)* Academia Press.

Published in:

Proceedings of the Fifth International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems (ACACES)

Queen's University Belfast - Research Portal:

[Link to publication record in Queen's University Belfast Research Portal](#)

General rights

Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy

The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.

Open Access

This research has been made openly available by Queen's academics and its Open Research team. We would love to hear how access to this research benefits you. – Share your feedback with us: <http://go.qub.ac.uk/oa-feedback>

Parallelization and Performance of an H.264 Video Encoder on the Cell B.E.

Michail Alvanos¹, George Tzenakis¹,
Dimitrios S. Nikolopoulos¹, and
Angelos Bilas¹

*Institute of Computer Science (ICS)
Foundation for Research and Technology – Hellas (FORTH)
100 N. Plastira Av., Vassilika Vouton, Heraklion, GR-70013, Greece*

ABSTRACT

Modern multicore processors with explicitly managed local memories, such as the Cell Broadband Engine (Cell) constitute in many ways a significant departure from traditional high performance CPU designs. A main issue in evaluating the features and limitations of modern multicore CPUs is appropriate workloads. In this work we first port an available H.264 video *encoder*, x264, on the Cell. x264 has been written for shared-memory multiprocessors that support coarse-grain parallelism. Porting x264 to the Cell requires extensive rewriting to avoid shared memory accesses and also to deal with the limited on-chip memory of the synergistic processing cores (SPEs). In our work we use TPC, an in-house runtime for the Cell. Our preliminary experimental results show speedup up to 270% compared to using only the PowerPC core (PPE). However, speedup may vary significantly depending on the input configuration. Going forward, a main challenge is to reduce application serial sections, especially as the number of cores increases in future multicore CPUs.

KEYWORDS: Cell; multicore; parallel applications; workload characterization; h.264; video encoding

1 Introduction

The Cell is a multicore processor [Hof05] equipped with one Power Processing Element (PPE) and eight high performance specialized Synergistic Processing Elements (SPEs). Both the PPE and the SPEs support SIMD extensions. Each SPE processor has a 128-bit datapath, 128 128-bit registers, and 256 KB of software-managed local store. The SPE is essentially an in-order, vector RISC platform, designed to achieve high performance using branch-eliminated vectorized code. The PPE and each SPE have a peak performance of 25.2 and 25.6 GFlops respectively, for a total of 230 GFLOPS.

¹E-mail: {alvanos,tzenakis,dsn,bilas}@ics.forth.gr

Video encoding and decoding is an important application for the embedded domain in general. Moreover, with increasing application requirements, e.g. HDTV, on video resolution and frame rates, both encoding and decoding are demanding given today's CPUs and systems. In particular, H.264 video encoding is a complex, multi-phase process that has received significant attention both at the algorithmic as well as the system level.

In this paper we are interested in examining the effort associated with providing efficient parallel video encoding on the Cell. We start from an existing parallel version of H.264 encoding, x264 [x26], that has been written for shared memory multiprocessors. x264 uses *frame* parallelism and allows multiple, coarse-grained threads to process different frames. On the other hand, the Cell provides neither a shared address space among SPEs nor is it appropriate for coarse-grained parallelism due to the limited per SPE local memory. Thus, porting x264 on the Cell requires extensive rewriting for: (a) using intra-frame parallelism by identifying finer-grained tasks and (b) explicitly managing communication and sharing among different concurrent tasks.

In our work we use an in-house runtime system, Tagged Procedure Calls (TPC), that has been written for the Cell. TPC is a task-based runtime. The programmer starts from a sequential program and identifies tasks that can execute concurrently. Tasks in TPC are portions of code and the associated data that will be accessed by this code. Thus, both control and data aspects of tasks are explicitly identified to the runtime. In our current implementation of TPC, the programmer can specify tasks in the form of procedure calls where all (global) data accesses are performed through arguments.

Next, we discuss the design of our parallel x264 video encoder on the Cell and present preliminary performance results.

2 Cell-based Parallel H-264 Design

H.264[Wie03] is a video compression standard. It is also known as MPEG-4 Part 10, or MPEG-4 AVC (for Advanced Video Coding). The X264 encoder, as a typical video encoder, consists of three main functional units: a temporal model, a spatial model, and an entropy encoder. An input video picture is divided in macroblocks (MBs), a corresponding 16x16-pixel region of a frame. The first step of the algorithm (temporal model) is to exploit the similarities between neighbouring video frames. In this step intra prediction also tries to find similarities between the current frame and neighbouring MBs. In the spatial model MBs are transformed using the Discrete Cosine Transform. The transformation outputs a set of coefficients that is next quantized. These values are combined with additional parameters and are converted into binary codes using either context-based adaptive variable length coding or context-based adaptive binary arithmetic coding.

2.1 Options of parallelism granularity

Although parallelization of x264 can occur at different granularities, the only viable approach for the Cell is parallelization at the macroblock level, where a single frame is being processed in parallel by all SPEs. The main reason for this is the limited on-chip memory of each SPE and the requirement for explicit communication management.

This type of parallelism requires satisfying macroblock dependencies. In H.264, motion vector prediction, intra prediction and the deblocking filter use data from neighboring MBs

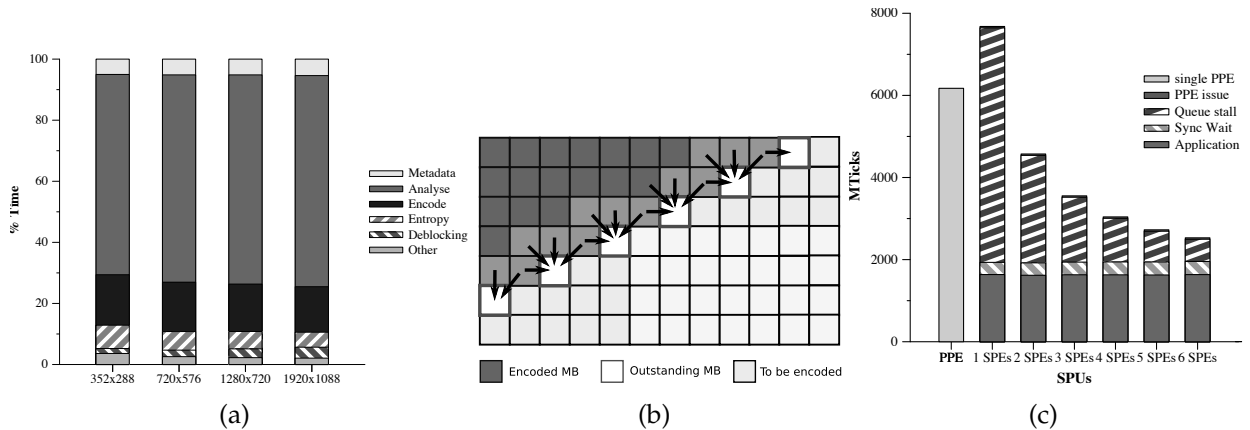


Figure 1: x264 execution time breakdown for different resolutions (left), 2D-Wavefront parallelism strategy (center), and TPC/x264 execution time breakdown for uneven multihexagon motion estimation (right).

defining a structured set of dependencies. Processing MBs in an antidiagonal-based manner (2D wavefront parallelism [vdTJG03]) satisfies all dependencies. However, there are two main disadvantages in this approach. First, entropy encoding must be applied serially to each frame due to its serial nature and second, the number of independent MBs varies during the encoding phase of each frame. The number of independent MBs in each frame depends on the resolution, for example in SD (720x576) and FHD (1920x1088) the max number of independent MBs are 23 and 60 respectively as reported in [CM08].

A variant of MB-level parallelism is slice-based parallelism. In this scheme, the encoder first decides the frame type and rate in a serial section of code. Then it splits the frame under consideration into several slices. The encoder waits until all slices have been processed and applies the deblock filter. There are two main drawbacks in this implementation: first, each slice reduces quality (adds extra bits for slice header and cabac contexts are reset) and scalability is limited significantly by the required serial code-sections.

To satisfy the intra-frame macroblock dependencies we use 2D-wavefront parallelism. Figure 1(b) shows the 2D-wavefront strategy applied to a frame, in diagonal manner. This approach provides a significant number of tasks that can be executed in parallel, although the number of independent MBs does not remain constant during the encoding of a single frame. The implementation was divided in two steps: First we replicated the data structures and restricted the global memory accesses of the kernels in PPE, and then with the help of TPC for the data transfers we ported the code to the SPE.

Finally, in our work, we vectorized some of the x264 kernels for the SPEs. We port most kernels responsible for motion estimation, sum of absolute differences, sum of absolute transformed differences, and pixel average. For the PPE, x264 already provides vectorized versions of the same kernels using the AltiVec extensions.

3 Evaluation

In our experiments we use a number of high definition (HD) video inputs taken from the DACI videobench [dac]. We run our encoder on a Playstation 3 game console system using the rally video sequence at 1280x720. PS3 has one Cell processor with 256 MBytes of off-

chip memory and allows the programmer to only access 6 of the 8 available SPEs in the Cell processor.

Figure 1(a) shows that most execution time is spent in the analysis and encoding phases of the code. We also observe that resolution does not affect significantly the percentage of execution time spent in analysis and encoding. The parallel section of the code currently accounts for about 85% of the serial execution time in the PPE, allowing for a maximum speedup of about 6. Figure 1(c) shows preliminary speedups and execution breakdowns. A slowdown of 0.85X against the initial version of the encoder is observed, when using one SPE. Using two or more SPEs we see a speedup between 1.29 and 2.7 (for 6 SPEs). Further improvements in speedup will require reducing the serial parts of the application (mainly the entropy encoding and the deblocking filter) and reducing inter-frame synchronization.

4 Conclusions

In this work we examine the design and performance of a parallel H-264 video encoder on the Cell. We start from an existing, thread-based parallel encoder, x264, and port it on the Cell by identifying the appropriate granularity for parallelism and by dealing explicitly with data placement and communication issues on the Cell. First, we note that the effort associated with achieving macroblock-level parallelism is significant, despite the availability of the original parallel version of the encoder. Our preliminary results show that our design achieves a speedup of up to 2.7 using six SPEs, compared to the PPE execution time. A main challenge for video encoding on future multicore CPUs is reducing the serial application sections and intra-frame synchronization, especially as the number of cores grows.

References

- [CM08] Mauricio Alvarez Ben Juurlink Alex Ramirez Cor Meenderinck, Arnaldo Azevedo. Parallel scalability of video decoders. Technical report, Universitat Politecnica de Catalunya (UPC), Delft University of Technology (TUD), Barcelona Supercomputing Center (BSC), 2008.
- [dac] <http://www.videobench.rd.tut.fi/>.
- [Hof05] H. Peter Hofstee. Power efficient processor architecture and the cell processor. In *HPCA*, pages 258–262, 2005.
- [vdTJG03] E.B. van der Tol, E.G.T. Jaspers, and R.H. Gelderblom. Mapping of h.264 decoding on a multiprocessor architecture. In *Image and Video Communications and Processing*, 2003.
- [Wie03] G.J.; Bjontegaard G.; Luthra A. Wiegand, T.; Sullivan. Overview of the h.264/avc video coding standard. volume 13 of *Circuits and Systems for Video Technology*, pages 560 – 576, July 2003.
- [x26] <http://www.videolan.org/developers/x264.html>.