Analyzing scalability limits of H.264 decoding due to TLP overhead

Mauricio Alvarez[1], Arnaldo Azevedo[2], Cor Meenderinck[2], Ben Juurlink[2], Andrei Terechko[3], Jan Hoogerbrugge[3], Alex Ramirez[1,4]
[1]Technical University of Catalonia (UPC), Barcelona, Spain
[2]Delft University of Technology, Delft, The Netherlands
[3]NXP, Eindhoven, The Netherlands
[4]Barcelona Supercomputing Center (BSC), Barcelona, Spain
Email: alvarez@ac.upc.edu, {azevedo, cor, benj}@ce.et.tudelft.nl, {andrei.terechko, jan.hoogerbrugge}@nxp.com, alex.ramirez@bsc.es

I. Introdution

Chip MultiProcessors (CMPs) are common in many areas of computing nowadays. In the pursuit of ever more performance ILP techniques were providing diminishing benefit while being power inefficient. Instead TLP has become the main driver for performance increase. Hardware nowadays provides a low amount of TLP in general although it is growing. It is generally expected that in the future many-cores with hundreds or thousands of cores will be feasible. To exploit such architectures there should be large amounts of TLP in the applications. We do recognize that this is a challenge, but do not consider it impossible. For example, in [1] we showed that H.264 video decoding exhibits large amounts of TLP. However, having a highly parallel architecture and application still does not guarantee parallel scalability, i.e., efficient exploitation of TLP on a high number of cores. In this work we analyze the TLP overhead of parallel H.264 decoding and how it limits the scalability. We have implemented two different parallelization strategies on three different architectures and analyzed the factors limiting the parallel scalability.

II. Background

H.264 is the best video coding standard in terms of compression and video quality. On the other hand, H.264 is also very computational intensive and has complex data dependencies. Therefore it has been regarded as hard to parallelize. A well know parallelization strategy (called 2D-Wave) was proposed in [2] and is fairly well scalable. Recently, the authors proposed the 3D-Wave strategy, which is even more scalable [1]. The 2D-Wave exploits parallelism among macroblocks (MBs) in one frame or slice. Roughly speaking, MBs on a diagonal from bottom-left to top-right are independent and can be processed in parallel. The amount of parallelism is limited to the size of the frame. The 3D-Wave also exploits parallelism among frames, even if frames depend on one another. The theoretical maximum parallelism for Full HD (FHD) resolution achieved by the 2D-Wave is 60 MBs while the 3D-Wave can achieve up to 7000 MBs in parallel. Results does not include entropy processing.

III. 2D-Wave on Altix

In order to verify the proposal of 2D-Wave and validate the theoretical and simulation results we have implemented it on a real multiprocessor machine. We modified the FFmpeg application for including the 2D-Wave parallelization and tested it on a SGI Altix which is a shared memory machine with a cc-NUMA architecture with 64 dual core IA-64 processors. Each one of the 128 cores works at 1,6 GHz, with a 8MB L3 cache and 533 MHz Bus, and the system has a total 512 GB RAM.

Three different scheduling algorithms were considered. The first one is static scheduling which assumes constant processing time and, based on that, it takes a predefined processing order of MBs. Second, a dynamic scheduling mechanism based on the task queue model in which the processing order is defined dynamically by the arriving of MBs into the task queue. And finally, dynamic scheduling was enhanced with tail submit optimization in which processors can decode some MBs directly without passing through the task queue.

The static scheduling approach fails to discover the available parallelism because it assumes a predefined processing order. Because of variable processing time threads spent most of its time waiting for others to finish. As way to eliminate this load-balancing problem dynamic scheduling was considered. Although this scheme solves the problem of the waits in the static scheduling it does not have a better scalability. This is due to the fact that in dynamic scheduling every MB has to be submitted and retired from the task queue and for doing that there is some thread synchronization overhead. As a result dynamic scheduling has a lower scalability than static scheduling.

As a way to reduce the overhead of thread synchronization tail submit was implemented. In this case MBs can be processed directly without being submitted and retired to/from the task queue. Tail submit has a better scalability compared to static and dynamic scheduling. A maximum speed-up of 9.5 (against the sequential version) was obtained for the version with tail submit and using 22 processors.

From the experimental analysis we conclude that 2D-Wave parallelization can provide significant performance benefits

but in order to be scalable it requires a corresponding fast and scalable synchronization and thread management support. Current interfaces for thread synchronization (like POSIX semaphores) entails a significant overhead which limits the performance of a fine grain parallelization like 2D-Wave.

IV. 2D-Wave on Cell BE architecture

We have implemented the 2D-Wave for the Cell BE architecture. We used our modified FFmpeg code and tested it on Cell Blades (QS-20) using IBM SDK 3.0.

We used the Cell SPUs as accelerators of MB decoding. That means that all the scheduling and thread synchronization is managed by the PPE. The PPE signals each SPU when a new MB is ready to be processed and then the SPU takes all the data by DMA, perform the decoding, write the results to main memory and finally sends a signal to the PPU informing that the work has been completed.

An extensive analysis of all the memory accesses was done in order to transform all of them to DMA commands and to implement prefetching for overlapping memory transfers with computation.

Based on the results from previous analysis on the SMP machine we selected the version with dynamic scheduling and tail submit. But we found that there is not a significant speed-up by using this parallelization on the Cell. Most of the time the SPU processors are waiting for the PPU master thread to give them work. This is the result of having all the scheduling and synchronization operations on the PPU which is not able to feed the all the SPUs. Additionally, the synchronization between PPU and SPUs was done using the mailbox modules available on the Cell, and this mechanism resulted to be very slow.

As a main conclusion from our work on the Cell, it is clear that the synchronization and scheduling should be done by the SPUs themselves. This require to use a different synchronization strategy not based on mailbox but on atomic instructions through the main memory. It is an open question if that SPU-centric synchronization model is going to provide the required performance for the 2D-Wave parallelization.

V. 3D-Wave implementation on an embedded multimedia multicore

An implementation of the 3D-Wave strategy is on a multicore architecture composed of NXP TriMedia TM3270 embedded processors will be found in [3]. Implementing the 3D-Wave turned out to be quite challenging. It required to dynamically identify inter-frame MB dependencies and handle their thread synchronization, in addition to intra-frame dependencies and synchronization. This led to the development of a subscription mechanism where MBs subscribe themselves to a so-called *Kick-off List* associated with the MBs they depend on. Only if these MBs have been processed, processing of the dependent MBs can be resumed.

We use this implementation to evaluate the effects of bandwidth requirements, intra-chip memory latency, amount of intra-core memory, and the influence of the parallelization support over scalability.

3D-Wave is 18% more bandwidth efficient than the 2D-Wave as it exploits inter-frame parallelism. However, frame scheduling can impact data locality. Depending on the resolution and the amount of allowed frames in flight the bandwidth efficiency can range from 15 to -8%.

We studied the effects of memory latency on the scalability of the 3D-Wave technique. The study was carried as the type of interconnection used, and the number of cores on the system contribute to increase the memory latency. The simulator does not simulate the communication protocol between L1 and L2. Instead, an average memory latency is set to delay the access of data request. Input sequences were decoded with the average memory latency set from 40 (original case) to 100 cycles. Results show that the scalability is just slightly affected by the increased memory latency. However, the performance lost can reach 40%.

To investigated the benefits of intra-core memory, we varied the amount of L1 data cache memory per core. The simulated amount of L1 cache was 16, 32, 64, 128, and 256KB. The resolution defined the optimal cache size. For SD and HD the 32KB cache had the best cost$\times$benefit ratio. For FHD the original size of 64KB had a better higher performance increase over the 32KB. None of the resolutions benefited for caches over 64KB.

A critical factor on multicore system is the parallelization support overhead, as presented previously on this work. This parallelization support includes mutex, locks, and thread/task management. We inserted artificial overheads of 10%, 20%, 30%, 40%, 50%, and 100% of the average MB decoding time on the task request function. The influence of the extra overhead was more or less severe depending on the resolution. For SD the scalability saturated at 22 while for the FHD performance drop was approximately 30%.

VI. Conclusions

In this work we have analyzed the factors that limit scalability of parallel H.264 decoding on different architectures. We evaluated the 2D-wave parallelization on a shared-memory cache coherent multiprocessor and on a heterogeneous multicore with software controlled local memories. Additionally we evaluated the 3D-wave parallelization for an embedded multimedia processor with cache coherent memories. For all this implementations we studied the effects thread management overhead. It is clear that the 3D-wave is more scalable than 2D-wave but bothof them work at the level of MB, which is very fine grained parallelization. Because of that, they require an efficient and scalable support in the architecture for thread shynchronization and task management.

References

[1] C. Meenderinck, A. Azevedo, B. Juurlink, M. Alvarez, and A. Ramirez, "Parallel scalability of video decoders," *Journal of Signal Processing Systems*, 2008.

[2] E. van der Tol, E. Jaspers, and R. Gelderblom, "Mapping of H.264 Decoding on a Multiprocessor Architecture," in *Proc. SPIE Conf. on Image and Video Communications and Processing*, 2003.

[3] A. Azevedo, C. Meenderinck, B. Juurlink, A. Terechko, J. Hoogerbrugge, M. Alvarez, and A. Ramirez, "Parallel H.264 Decoding on an Embedded Multicore Processor," in *(To Appear) Proc. Hipeac Conference*, 2009.