Designing Real-Time H.264 Decoders with Dataflow Architectures

Youngsoo Kim Suleyman Sair

Department of Electrical and Computer Engineering, NC State University

{ykim12,ssair}@ ece.ncsu.edu

ABSTRACT

High performance microprocessors are designed with general-purpose applications in mind. When it comes to embedded applications, these architectures typically perform control-intensive tasks in a System-on-Chip (SoC) design. But they are significantly inefficient for data-intensive tasks such as video encoding/decoding. Although configurable processors fill this gap by complementing the existing functional units with instruction extensions, their performance lags behind the needs of real-time embedded tasks. In this paper, we evaluate the performance potential of a dataflow processor for H.264 video decoding. We first profile the H.264 application to capture the amount of data traffic among modules. We use this information to guide the placement of H.264 modules in the WaveScalar dataflow architecture. A simulated annealing based placement algorithm produces the final placement aiming to optimize the communication costs between the modules in the dataflow architecture. In addition to outperforming contemporary embedded and customized processors, our simulated annealing guided design shows a speedup of 13% in execution time over the original WaveScalar architecture. With our dataflow design methodology, emerging embedded applications requiring several GOPS to meet real-time constraints can be drafted within a reasonable amount of design time.

Categories and Subject Descriptors

C.1.3 [Other Architecture Styles]: Dataflow architectures

General Terms: Design, Performance

Keywords: Dataflow architecture, H.264, WaveScalar

1. INTRODUCTION

The specialized needs of embedded applications create several problems when running on general-purpose processors. Typically critical functions in embedded applications cannot be accelerated due to a mismatch between the application and the default instruction set architecture of the processor. As a result, embedded applications take many cycles to execute on general-purpose processors. Secondly, many embedded applications cannot use general-purpose or DSP processors because such an implementation is not feasible due to the high power consumption and IP costs. Consequently, the advent of new video encoding/decoding standards such as H.264 has created the need for an ASIP (Application Specific Instruction Processor) solution to accelerate bottleneck functions in order to meet real-time requirements (e.g. 15 frames per second on low bit rate communication lines).

Extending the instruction set can significantly accelerate the overall performance, especially for data-dominated applications that represent many of contemporary embedded handheld SoCs. Moreover, automatic synthesis of configurable processor cores, and the associated software tool chain - including retargetable compilers and simulators, maximizes the applicability of configurable processor cores. However, both in literature surveys and our limit study of a typical configurable processor, we found that instruction extension approaches provide speedups in the 30-50% range, far short of the speedups needed to realize future embedded applications with low-end embedded processors [1].

Several researchers have proposed to overcome these problems. One approach is using coarse grain reconfigurable processor architectures such as Morphosys [2] and ADRES frameworks [3]. Another approach is the use of dataflow architectures in order to decentralize the processor architecture. One of the advantages of dataflow architectures is that they can be used as an alternative to RTL design and be extremely instrumental in reducing the turn around time of current design methodologies. Because the specifications of these emerging embedded applications such as MPEG4, H.263, H.263+, and H.264 are constantly changing, implementation on a generic fabric has many advantages in the long run. But very few research efforts propose the design methodology for a realistic embedded application on a dataflow substrate.

We present a design methodology to implement a real-time H.264 video-decoding application on a dataflow processor in this paper. First, we present a limit study of extending the ISA of a configurable processor to examine the available processing power for emerging embedded applications. Then, we discuss our design methodology, which uses the data transfer matrix of the H.264 application to create a mapping of application modules to computation units on the WaveScalar dataflow architecture [11]. Since H.264 is extremely data-dominated, we chose a dataflow architecture to reduce the amount of data transfer between modules of the application.

2. RELATED WORK

Many researchers have investigated using configurable architectures and extending the basic instruction set to speedup the execution of specific applications. In these designs, we typically start out with a parameterized processor and its basic instruction set. We then use an architectural description language to generate the datapaths or the functional units needed for the extension instructions. To complement the new hardware, we need to create the necessary software tools such as the compiler, instruction set simulator (ISS), debugger and assembler. This completes the design cycle. With the help of automatic tool chains, designers can develop an application specific processor in a short amount of time. Even though these tools can significantly shorten one iteration of the design cycle, many iterations are needed to find the extension candidates that result in sufficient acceleration of the application [4]. One has to rely on simulation to find the performance bottlenecks in the application so that they can be mapped to extension instructions. This process typically involves using very slow ISSs [5]. Moreover, the automatic synthesis of functional units typically produces large register files because most extended instructions are SIMD style instructions.

One technique for automatic extraction of the extended instruction candidates is proposed in [6]. Unlike traditional two-input, one-output matching, they identify Multiple Input, Single Output (MISO) operations and cluster them into one custom instruction. This can be effective for embedded applications that have high computation needs (e.g. cryptography), but most data-intensive programs have very little arithmetic code. As a result, the arithmetic portion of the application is not a bottleneck in program execution. Another instruction set extension approach is the CCA (Configurable Compute Accelerator) [7]. In the CCA approach, original application is profiled to find the most frequently executing traces. Using these traces, candidate subgraphs for extension instructions are discovered. Next the compiler generates a binary that takes advantage of these CCA instructions. The CCA hardware is a grid of ALUs with multiple copies of different functional units. Although CCA aims to reduce the hardware design and verification time for different applications, modern design automation tools for configurable processors are fairly successful and widely used in industry [8][9][10]. Because most configurable processors are designed in a parameterized manner, the automatic synthesis of whole processor including the customized functional units takes a relatively short time to complete [9].

Meanwhile, commercial extensible processors have long been available in the industry. Tensilica announced their Xtensa V processor, which is based on a 32-bit RISC processor architecture in 2002[9]. They extended the instruction set with user-defined instructions. From the TIE (Tensilica Instruction Extension) description of new instructions, a new datapath or functional unit is added to the processor and a new integrated development environment that supports the new instruction is generated.

In addition to instruction set extension approaches, many coarse grain reconfigurable architectures have been proposed. Morphosys has a tinyRISC processor and a reconfigurable ALU matrix to extend the functional units [2]. The ADRES framework tightly couples a reconfigurable ALU matrix with a VLIW processor. The data kernel is executed on the reconfigurable FUs and the rest of the code is executed on the VLIW processor [3]. While ADRES aims to improve the non-kernel code execution by adopting a VLIW processor instead of a RISC processor, it is not clear how much the whole application can benefit from speeding up the control intensive part of the code. Moreover, neither technique investigates intelligent mapping strategies to reduce the communication overhead between the host processor and the reconfigurable matrices.

Recently, the WaveScalar dataflow architecture was proposed [11]. WaveScalar has a dataflow instruction set such that each instruction executes in place in the memory system and explicitly communicates with its dependents in dataflow fashion. WaveScalar architectures cache instructions and the values they operate on in a WaveCache, a simple grid of "ALU-in-cache" nodes. By co-locating computation and data in physical space, the WaveCache minimizes long wire, high-latency communication. For an efficient implementation of the H.264 video decoding standard, WaveScalar was selected as a substrate because it is capable of significantly reducing the amount of data transfer in H.264. In addition, due to the highly distributed nature of the application, WaveScalar provides a good model to speedup most of the performance bottlenecks.

3. H.264 STANDARD

Before discussing the configurable processor and dataflow implementations of H.264, let us briefly summarize the standard. H.264 is the video encoding/decoding standard that is replacing the current MPEG4 video standard [12]. The new standard provides both enhanced compression efficiency over existing video coding standards and a network friendly video representation. In its video coding layer, some of the important enhancements include the use of a small block-size, exact match transform, adaptive in-loop deblocking filter and motion prediction capabilities. A detailed block diagram of the H.264 decoder is presented in Fig. 1. After receiving the data from NAL (Network Adaptation Layer), the data is processed by the entropy decoder. Next, the IT/IQ (Integer transform/Inverse quantization) block is used to generate the reference frame data which will be added to the reference frame image or intra-predicted image based on its header information. Then, the original image in reconstructed through the deblocking filter. The H.264 baseline profile used in this project is a simple implementation that supports intra and inter-coding, as well as entropy coding with context-adaptive variable-length coding. The reference C source code is built by pruning the TML reference model. Extra functionality beyond the selected H.264 profile was removed from C code. Therefore, the C model that we use has been optimized at the source code level.

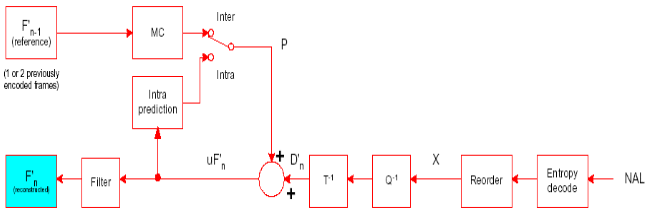


Fig. 1. H.264 Decoder

There are several on-going research efforts analyzing the complexity of H.264 video decoding. The complexity of the algorithm directly affects the cost-effectiveness of an H.264 implementation and hence the final success of the standard. The published results show that H.264 standard aims at improving the compression efficiency up to 50% compared to existing standards. However, the encoder complexity increases by a factor of 2 when compared to MPEG4 and with a factor of three for the decoder [13]. Finally, it is impossible to realize the decoder completely in software on an embedded processor at this time. This is the reason the H.264 application is selected as a target of our design methodology. The H.264 is the most complex algorithm in video decoding area. Unlike other areas such as cryptography, the embedded video coding application must be run at real-time speed. Therefore, it would be ideal if we can achieve sufficient acceleration of the at a moderate hardware cost [14]. On the other hand, this kind of applications is also unsuitable for a hardware only implementation because the standards are always evolving and changing.

4. CONFIGURABLE PROCESSOR LIMIT STUDY

In this section, we present a limit study on the performance potential of a typical configurable processor. The goal of this study is to get a good grasp of the processing power of current configurable processors when we add new instructions to the ISA. In Section 5, a currently available embedded processor (an ARM9) will be compared to this configurable processor and our dataflow architecture in their ability to meet the real-time requirements for H.264.

We configure the H.264 reference code to process 15 frames of video data per second. We then profile the application with the GNU gprof toolset. Next, we analyze the longest executing functions in the profile to determine which instructions to accelerate. The final step involves rewriting the application binary where we use inline assembly to insert the mnemonics for the newly defined instructions.

As is common in configurable processor research [15][16], we simulated the original and the rewritten binaries in the SimpleScalar ARM simulator [18]. We modified SimpleScalar instruction definition files to correctly recognize and simulate new instructions on the new functional units.

The profiling results of the decoder reference software are presented in Fig. 2. As expected, MC (Motion Compensation) block is the most performance demanding part of the decoder.

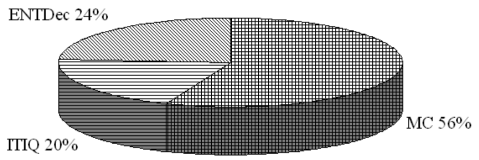
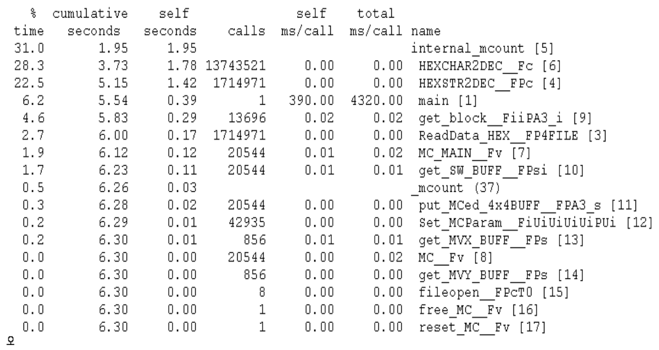


Fig. 2. H.264 decoder gprof profiling results

In addition to global profiling, we also profile MC, ITIQ, ENTDEC blocks separately. The profiling results for the MC block are presented in Fig. 3. After removing GNU gprof instrumentation functions (e.g. internal_mcount, _mcount), along with the driver functions and the stubs needed for simulating image input (e.g. HEXCHAR2DEC, main, get_block, ReadData_HEX), MC_MAIN and get_SW_BUFF remain as the main bottlenecks. MC_MAIN performs the difference calculation of nearby images and get_SW_BUFF reads in the motion block for processing.



```
  %  cumulative   self              self    total
 time   seconds   seconds    calls  ms/call  ms/call name
 31.0      1.95     1.95                             internal_mcount [5]
 28.3      3.73     1.78 13743521     0.00     0.00  HEXCHAR2DEC__Fc [6]
 22.5      5.15     1.42  1714971     0.00     0.00  HEXSTR2DEC__FPc [4]
  6.2      5.54     0.39        1   390.00  4320.00  main [1]
  4.6      5.83     0.29    13696     0.02     0.02  get_block__FiiPA3_i [9]
  2.7      6.00     0.17  1714971     0.00     0.00  ReadData_HEX__FP4FILE [3]
  1.9      6.12     0.12    20544     0.01     0.02  MC_MAIN__Fv [7]
  1.7      6.23     0.11    20544     0.01     0.01  get_SW_BUFF__FPsi [10]
  0.5      6.26     0.03                             _mcount (37)
  0.3      6.28     0.02    20544     0.00     0.00  put_MCed_4x4BUFF__FPA3_s [11]
  0.2      6.29     0.01    42935     0.00     0.00  Set_MCParam__FiUiUiUiUiPUi [12]
  0.2      6.30     0.01      856     0.01     0.01  get_MVX_BUFF__FPs [13]
  0.0      6.30     0.00    20544     0.00     0.02  MC__Fv [8]
  0.0      6.30     0.00      856     0.00     0.00  get_MVY_BUFF__FPs [14]
  0.0      6.30     0.00        8     0.00     0.00  fileopen__FPcT0 [15]
  0.0      6.30     0.00        1     0.00     0.00  free_MC__Fv [16]
  0.0      6.30     0.00        1     0.00     0.00  reset_MC__Fv [17]
```

Fig. 3. MC block profile

Based on profiling results, new instructions are created for reducing the execution time spent in MC_MAIN and get_SW_BUFF functions. In general, a new instruction falls into one of the following categories:

- SIMD instruction – The instruction performs the same operation on multiple data items in parallel. These operations are prevalent in video applications and thus can provide significant performance improvements with specialized SIMD instructions.
- Combined instruction – When multiple operations are applied to single data item sequentially, the new instruction can combine these into one instruction. Unlike compiler optimizations or other software approaches that aim to reduce the instruction count locally, the search for combining instructions takes a global approach. In the most extreme case, the whole application can be replaced with a single instruction. As an example, the showbits function which takes the most execution cycles in the entropy decoding block (ENTDEC) can be replaced with one instruction using relatively few hardware resources.

Fig. 4 shows the pseudo code of the function that calculates the difference between pixels in each motion block and accumulates the differences for motion compensation. We will replace this whole loop with one instruction. The new SAD (Sum of Absolution Difference) instruction will be used in the subsequent experiments instead of this loop.

```
for y_s                /* motion vectors in search area */
 for x_s
  for y_p              /* pixels in macro-block            */
   for x_p
    sad[x_s][y_s] +=
        abs(prev_image[x_s+x_p][y_s+y_p]
             - curr_image[x_p][y_p]);
```

Fig. 4. Pseudo code of SAD function

We compiled the H.264 software with glibc 2.1.3 using the arm-linux library. In our simulations, we used a StrongARM 110 configuration. As our benchmarks, we utilize a hand-coded unit test bench for speed up calculation and the H.264 reference software with 15 frames of image data.

```
for (row = 0; row < 16; row++) {
        Pixelrow row1, row2;
        /* do SAD on all 16 pixels in row at once: */
        row1 = *((Pixelrow *) im1_p);
        row2 = *((Pixelrow *) im2_p);
        SAD(total,row1, row2);
        /* point to first pixel in next row of block in image */
        im1_p += numcols;
        im2_p += numcols;
    } /* row loop */
```

Fig. 5. SAD2 loop code for unit testing

The unit test bench SAD1 (original code) and SAD2 (modified code with new instruction) were written by hand. The code for SAD2 is presented in Fig. 5. Execution time results for the unit test and H.264 are presented in Fig. 6. SAD2 is approximately 15 times faster than SAD1 code, because the extended instruction performs sixteen sum of absolute difference calculation in one clock cycle. The H.264 with 15 frames of video data shows a speedup of 2.9. In addition to SAD, 11 additional instructions were added to the ISA to achieve this speedup. In all, we selected the top four candidate instructions from each of the three H.264 modules. Therefore, this study is a good estimate for the performance of current configurable processors.

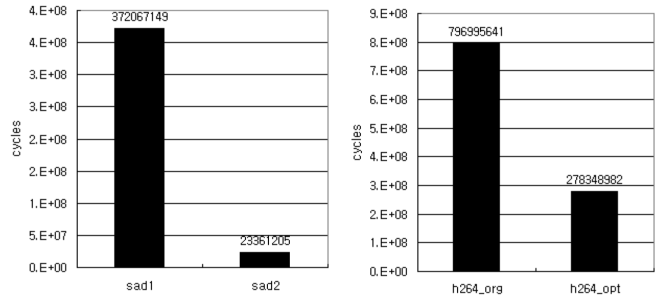


Fig. 6. Unit test and H.264 decoder simulation results with the new SAD instruction

5. H.264 on a DATAFLOW ARCHITECTURE

As mentioned in Section 2, WaveScalar is an alternative to superscalar designs for scalable, low-complexity, and high-performance processors with a dataflow instruction set architecture and execution model. To guarantee a low complexity design, a distributed architecture was modeled instead of centralized processing units in WaveScalar. WaveScalar's execution model relies on WaveCache processing elements that are comprised of a grid of ALUs-in-cache. WaveScalar instructions execute in their assigned WaveCache and send their results to their dependent's WaveCache. By co-locating computation and data in physical space, the WaveCache minimizes long wire, and thus high-latency, communication. Neighboring WaveCaches are arranged into 4 by 4 clusters. Each processing element has functional units, input and output queues and control logic to communicate. Each element has an instruction queue that is capable of storing eight instructions and the corresponding store buffers. The processing elements can communicate by using a shared bus in the cluster. Among the clusters, the WaveScalar uses a hop-by-hop routing policy where each link is traversed in one cycle [11].

For multimedia embedded applications like H.264 and MPEG4, acceleration typically implies approaches such as instruction-level parallelism and SIMD parallelization. These approaches are performed at a fine granularity. But coarse grain parallelism has more potential if the application is data-dominated. The entropy decoding block extracts the coded bit stream and forwards the motion vectors and quantized values to the integer transform block. Also, the motion vectors and prediction modes must be passed to motion compensation.

To use the inherent task level parallelism in H.264, we profiled the data transfer patterns to guide the placement of WaveCache blocks. In our profiling setup, the H.264 reference software was used and MB (Motion Block) data structure was the major profiling target. The parts of the application source code that operate on the motion block related C-struct were hand-instrumented to count the number of data transfers. The total amount of data transfer was calculated by multiplying the access counter with the size of each access. Note that it is fairly straightforward to incorporate this profiling code with a dynamic instrumentation system such as ATOM [19]. The data transfer matrix for H.264 is depicted in Fig. 7, with each node denoting a function block. Each edge represents the number of data transfers between blocks.

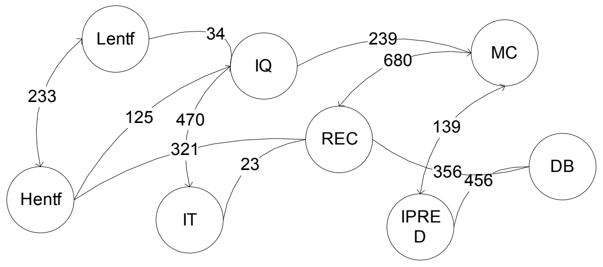


Fig. 7. Data Transfer Matrix of H.264

We use the transfer matrix to improve the default WaveScalar assignment of instructions to WaveCache clusters in our methodology. In the default WaveScalar architecture, WaveCache placement is performed by placing each producer and consumer as close as possible. In our methodology, we use a simulated annealing based placement algorithm. The slicing model uses WaveCache module connectivity information in minimizing communication costs. The objective of this algorithm is to set the relative position of function blocks such that the blocks contributing most to the overall data movement are placed as neighboring WaveCaches. Given this objective, the problem can be described as follows:

Minimize cost = $\sum \alpha_i * \text{netlength}_i$

where, i = each edge in the transfer matrix,

α_i = weight of edge i,

netlength_i = half perimeter net length of the nodes connected by i

We use the transfer matrix computed from the data usage profile to place blocks with large weights as close as possible. While net length does not correspond to an actual wire length, it represents a measure of block connectivity.

When we compare the original WaveScalar placement to our approach on the H.264 application, simulation results show that execution time is reduced by 13%. The total communication among processing elements is reduced by 25% with this transfer matrix based placement. In Table 1, nearby communication refers to the sum of the number of messages passed within a 4x4 cluster of processing elements. Fig. 8 depicts the communication amount among blocks of various sizes as a percentage of the total communication. We can see that our placement algorithm increases communication within a 4x4 block by almost 40% while reducing 8x8 communications by more than 60%.

Table 1. Execution cycles comparison

	Original	w/ transfer matrix
Execution cycles	165.95 Mcycles	144.49 Mcycles
Nearby communication	8182082	10992009

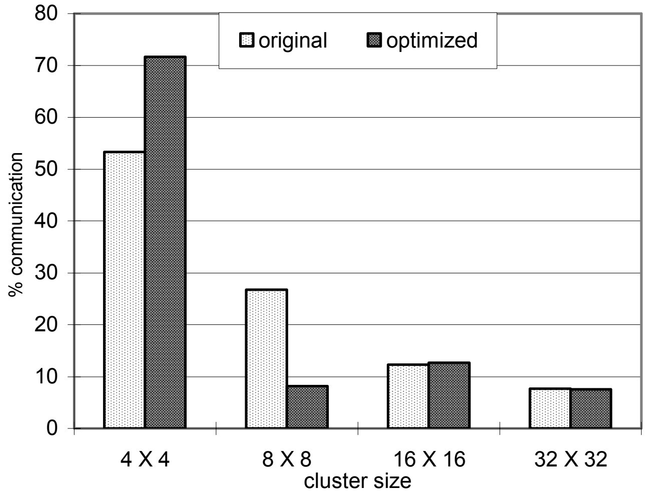


Fig. 8. Nearby communication histogram

We also analyzed a different application (the MPEG4 kernel) using our methodology. Our profiler produced the transfer matrix shown in Fig. 9. In terms of execution time, the original placement finished MPEG4 in 17.3 million cycles while the optimized version completes in 15.5 million cycles. This amounts to a 10% speedup.

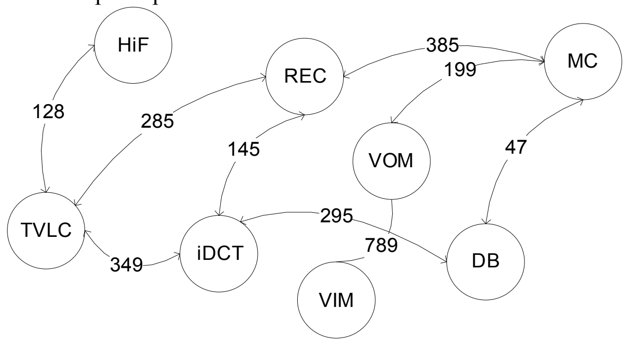


Fig. 9. Data Transfer Matrix of MPEG4

We can use these experimental results to investigate the real-time feasibility of a H.264 decoder implemented using different models. In our analysis, we assume that the processing of 15 frames in one second achieves real-time realization because H.264 is usually used in the low bit rate mobile environment. To our knowledge, there is no embedded processor that can handle this video application in real-time currently. The execution times of H.264 on various platforms are shown in Table 3. The ARM9TDMI processor is benchmarked with the instruction set simulator that ships with the ARM developer suite. We observe that in order to process H.264 in real-time, a clock speed above 300 MHz is needed. Note that this is only available in high-end embedded processors due to power constraints. When we compare the remaining two options, we see that the implementation on the dataflow architecture is feasible because of its reduced runtime. Because typical contemporary embedded processors operate up to a frequency of 200MHz, the amount of speedup is crucial in determining feasibility [9].

Next we compare the area of the designs we evaluate. All the area estimations presented in Tables 2 and 3 are based on a 0.18 um technology. The baseline configurable processor is the 5-stage RISC processor design provided by LisaTek and is synthesized with the Synopsys Design Compiler [20]. This synthesis yielded a 1.0 mm^2 area for the core. The area for extended instructions is computed by synthesizing the automatically generated HDL netlists from the LISA architecture description language. The estimated area results provided in Table 2 are calculated from the gate counts as well as routing utilization and additional resources.

Table 2. Configurable processor area

Extended instruction	Gate counts	Estimated Area
mc_ext	25,000	0.43 mm^2
iluma	7,803	0.13 mm^2
itrans	18,034	0.31 mm^2
showbits	14,837	0.25 mm^2
code_bitstr	12,023	0.21 mm^2
Rest	8,000	0.14 mm^2

In our dataflow design, we used 1024 WaveCache blocks for H.264 and each block has 8 instructions each of which is 4 bytes. Thus, in order to calculate the area of the dataflow processor, we used the die area of a 32KB, 32-byte block size, 1-way I-cache estimated by CACTI [17].

Table 3. H.264 real-time realization comparison

H.264	ARM9 TDMI	Configurable processor	Dataflow w/ transfer
Execution cycles	728.12 Mcycles	278.34 Mcycles	144.49 Mcycles
Estimated die area	1.16 mm^2	2.47 mm^2	3.2 mm^2

When we analyze the total die area results shown in Table 3 we see that dataflow architectures have a definite advantage in terms of performance per area. This is an important factor when one is considering incorporating accelerators into single SoCs.

6. CONCLUSIONS

Emerging embedded applications strain current embedded processors when it comes to real-time realization in a low power setting. In this paper, an efficient design methodology is proposed for one such application, the H.264 video decoder. To exploit the task parallelism inherent to the H.264 application, we calculated the data transfer matrix of this application in our methodology. We then mapped the H.264 modules onto the WaveScalar dataflow architecture with the help of a simulated annealing-based placement algorithm that uses the data transfer matrix to place blocks that communicate a lot close to one another. The experimental results show that this design method is extremely helpful in data-dominated applications. We also looked at the feasibility of implementing H.264 with an ARM9 embedded processor as well as a configurable processor and found that these approaches lack the sheer compute power demanded in this application. Our design methodology enables designers to implement data-dominated applications with real-time constraints in a short amount of time.

7. REFERENCES

[1] ITRS 2003-2018 Roadmap – System Functional Requirements For Handheld Wireless Low Power SoC

[2] H. Singh, Lee Ming-Hau, Lu Guangming, F. J. Kurdahi, N. Bagherzadeh and E. M. Chaves Filho, MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications, IEEE Transactions on Computers, Volume 49, pp. 465 - 481, 2004.

[3] B. Mei, S. Vernalde, D. Verkest and R. Lauwereins, Design methodology for a tightly coupled VLIW/reconfigurable matrix architecture: a case study, in Proc. DATE, pp. 1224-1229, 2004.

[4] A. Hoffmann, A, T. Kogel, A. Nohl, G. Braun, O. Schliebusch, O. Wahlen, A. Wieferink, H. Meyr, A novel methodology for the design of application-specific instruction-set processors (ASIPs) using a machine description language, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume 20, Issue 11, pp. 1338 – 1354, 2004.

[5] I. Park, S. Kang and Y Yi, Fast cycle-accurate Behavioral Simulation for Pipelined Processors Using Early Pipeline Evaluation," International Conference on Computer Aided Design, pp. 138-141, Nov, 2003

[6] K. Atasu, L. Pozzi, and P. Ienne, Automatic Application-Specific Instruction-Set Extensions under Microarchitectural Constraints, In the Proceedings of 40th DAC Design Automation Conference, Los Angeles, June 2003.

[7] N. Clark, M. Kudlur, H. Park, S. Mahlke, and K. Flautner, Application-Specific Processing on a General-Purpose Core via Transparent Instruction Set Customization, International Symposium on Microarchitecture (MICRO-37), pp. 30-40, December 2004.

[8] C. Rowen and S. Leibson , Flexible Architectures for Engineering Successful SOCs , In the Proceedings of 41st Conference on Design Automation Conference, pp. 692-697. 2004.

[9] Tensilica web page, http://www.tensilica.com/

[10] ARC website, http://www.arc.com

[11] S. Swanson, K. Michelson, A. Schwerin and M. Oskin, WaveScalar In the 36th Annual International Symposium on Microarchitecture (MICRO-36), December 2003

[12] H.264 TML Model, http://bs.hhi.de/~suehring/tml/

[13] S. Saponara and C. Blanch, K. Denolf and J. Bormans, The JVT Advanced Video Coding Standard: Complexity And Performance Analysis On A Tool-by-tool Basis, ICIP Conference, 2002.

[14] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer and T. Wedi, Video coding with H.264/AVC: tools, performance, and complexity, IEEE Circuits and Systems Magazine, Vol. 4, Issue 1, pp. 7-28, 2004.

[15] L. Pozzi, M. Vuletic, and P. Ienne, Automatic topology-based identification of instruction-set extensions for embedded processors. In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, page 1138, Paris, March 2002.

[16] N. Pazos, A. Maxiaguine, P. Ienne, and Y. Leblebici. Parallel modelling paradigm in multimedia applications: Mapping and scheduling onto a multi-processor system-on-chip platform. In Proceedings of the International Global Signal Processing Conference, Santa Clara, Calif., September 2004.

[17] CACTI web page, http://research.compaq.com/wrl/people/jouppi/CACTI.html

[18] D. Burger and T.M. Austin. The SimpleScalar Tool Set, Version 2.0. Technical Report 1342, Computer Sciences Dept., University of Wisconsin-Madison, 1997.

[19] A. Srivastava and A. Eustace, ATOM: A system for building customized program analysis tools**.** In Proceedings of the Conference on Programming Language Design and Implementation, pages 196--205. ACM, 1994.

[20] CoWARE LisaTek Processor Designer Manual.