System Design using Kahn Process Networks: The Compaan/Laura Approach

Todor Stefanov Claudiu Zissulescu Alexandru Turjan Bart Kienhuis Ed Deprettere
Leiden Embedded Research Center
Leiden Institute of Advanced Computer Science
Leiden University, The Netherlands
{stefanov,zissules,aturjan,kienhuis,edd}@liacs.nl

ABSTRACT

New emerging embedded system platforms in the realm of high-throughput multimedia, imaging, and signal processing will consist of multiple microprocessors and reconfigurable components. One of the major problems is how to program these platforms in a systematic and automated way so as to satisfy the performance need of applications executed on these platforms.

In this paper, we present our system design approach as an efficient solution to this programming problem. We show how for an application written in Matlab, a Kahn Process Network specification can automatically be derived and systematically mapped onto a target platform composed of a microprocessor and an FPGA. Furthermore, we illustrate how the mapping approach is applied on a real-life example, namely an M-JPEG encoder.

1. INTRODUCTION

New emerging embedded system platforms in the realm of high-throughput multimedia, imaging, and signal processing consist of multiple microprocessors and reconfigurable components. To satisfy the performance needs of tomorrows applications, these emerging platforms leverage task-level parallelism, i.e., the microprocessors and the reconfigurable components run concurrently. To execute an application on these platforms, the platforms have to be programmed, which implies writing software for the microprocessors using languages like C and writing hardware descriptions using languages like VHDL to configure the reconfigurable components.

To use the concurrency available in the platforms, we need to program them in a way that we exploit *distributed control* and *distributed memory*. Distributed control means that the individual components on a platform can proceed autonomously in time without much interference with other components. Distributed memory means that the exchange of data is contained in the communication structure between individual components and not pooled in a large global memory. Although distributed memory and control are key requirements to take advantage of the new emerging platforms, we observe that imperative programming languages like C, Java, or Matlab are still the preferred way to write applications that execute on these platforms. The imperative model of computation makes it easy to reason about a program as only a single thread of control needs to be considered. Also, memory is global and all the data comes from the same memory source. But precisely the single memory and single thread of control in the imperative model of computation are contradictory to the need for distributed control and memory. Therefore, programming these new platform is a very tedious, error prone, and time consuming process.

Instead, we believe that a much more appropriate model of computation is the Kahn Process Network model as it inherently expresses applications in terms of distributed control and memory. As said before, most applications are written in an imperative model of computation. To facilitate the migration from an imperative application to a KPN specification, we have developed the COMPAAN/LAURA approach. This approach allows parts of an application written in a subset of Matlab to be converted automatically to KPNs (COMPAAN). This conversion is fast and correct by construction. The obtained processes in a KPN can subsequently be mapped either in software or on hardware (LAURA).

In this paper, we present our system design approach that is centered around exploiting the Kahn Process Network model characteristics. We present our approach by illustrating how we map an M-JPEG application written in Matlab onto a target architecture that consists of a CPU and an FPGA. Our design approach consists of two major steps. In the first step, we convert the Matlab specification of the M-JPEG to a KPN specification. In the second step, we map one process in hardware on the FPGA whilst the remaining processes are mapped in software on the CPU. Before we explain our approach in more detail, we first look at the KPN model and its specific characteristics.

1.1 Kahn Process Networks

The KPN model of computation [1][2] assumes a network of concurrent autonomous processes that communicate in a point-to-point fashion over unbounded FIFO channels, using a *blocking-read* synchronization primitive. Each process in the network is specified as a sequential program that executes concurrently with other processes. A KPN has the following favorable characteristics:

- The KPN model is deterministic, which means that irrespective of the schedule chosen to evaluate the network, always the same input/output relation exists. This gives us a lot of scheduling freedom that we can exploit when mapping processes to hardware or software.
- The inter-process synchronization is done by a blocking read. This is a very simple synchronization protocol that can be realized easily and efficiently in hardware and software.
- Processes run autonomously and synchronize via the blocking read. When mapping processes on hardware like an FPGA, you get autonomous islands on the FPGA that are only synchronized via blocking reads.
- As control is completely distributed to the individual processes, there is no global scheduler present. As a consequence, partitioning a KPN over a number of reconfigurable components or microprocessors is a simple task.
- As the exchange of data has been distributed over the FIFOs, there is no notion of a global memory that has to be accessed by multiple processes. Therefore, resource contention does not occur.

Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'04)
1530-1591/04 $20.00 © 2004 **IEEE**

1.2 Compaan/Laura

Our system design approach is centered around the COMPAAN and LAURA tools which we have developed to facilitate this approach. The COMPAAN compiler, introduced in [3] and further developed in [4, 5], fully automates the transformation of Matlab code into Kahn Process Network (KPN) specifications. The applications COMPAAN can handle, have to be specified as parameterized static nested loop programs, which is a subset of the Matlab language. COMPAAN consists of three tools. The first tool transforms the initial Matlab code into single assignment code (SAC), which resembles the *dependence graph* (DG) of the initial nested loop program. The second tool converts the SAC into a Polyhedral Reduced Dependence Graph (PRDG) data structure, which is a compact mathematical representation of the DG in terms of polyhedra. The third tool converts the PRDG into a process network by associating a process with each node of the PRDG. The parallel processes communicate with each other according to the data-dependency given in the DG.

LAURA [6] maps a KPN specification onto hardware, for example, FPGAs. The LAURA tool operates as a back-end for the COMPAAN compiler. First, the KPN specification is converted into a functionally equivalent network of virtual processors, called *hardware model*. This is a platform independent step as no information on the target platform is taken into account. Second, platform specific information is added as well as IP cores to this hardware model leading to a network of synthesizable processors. Finally, the hardware model is converted into synthesizable VHDL code.

1.3 Related Work

Mapping applications like MPEG and JPEG codecs onto a target architecture consisting of a CPU and an FPGA has been the central question in Hardware/Software codesing in the last decade [7]. Researchers have already mapped successfully multi-media applications on such kind of platforms in a systematic way. The retargetable framework `Nimble` [8] and the work presented in [9] automatically compiles system-level applications specified in C onto a target architecture of a combined CPU and FPGA. However, the compiler only exploits instruction-level parallelism (ILP) in loops but not task-level parallelism. Loops are executed purely sequentially according to their original C specification even if mapped onto the FPGA for acceleration. In [10, 11], reconfigurable logic is used as a co-processor attached to a CPU. The co-processor is typically used to speeds-up certain instructions of the CPU.

All of the related work, mentioned above, exploits only ILP in loops mapped onto an FPGA that runs mutually exclusive with the CPU. In the work presented in this paper, however, we show a systematic and automated approach to map an application onto a CPU and an FPGA in such a way that the CPU and the FPGA run *concurrently*, exploiting *task-level* parallelism.

Some recent efforts in mapping applications onto a CPU connected to reconfigurable logic (FPGAs) exploiting task-level parallelism has led to approaches that are somehow related to the approach presented in this paper. Gokhale *et al.* [12] have developed a compiler that takes stream-based application specified in Streams-C and generates synthesizable hardware for FPGAs and a multi-threaded software program for the control CPU. The Stream-C programming task-level model is the CSP [13] model of computation. The work in [14] also presents an approach to map applications specified as CSPs onto a platform that consists of a CPU and FPGA. Conceptually, our approach differs from these approaches in the sense that we use the Kahn Process Network (KPN) model which specifies more naturally and efficiently (compared to CSP) the task-level parallelism in stream-based applications. The UC Berkeley's project SCORE [15] has developed a stream-based compute model which virtualizes reconfigurable computing resources (compute, storage, and communication) by dividing a computation up into fixed-size "pages" and time-multiplexing the virtual pages on available physical hardware. The specific language TDF is used to specify applications using the SCORE's model. This stream-based model is similar to the KPN model we use in this paper.

In the three approaches [12, 14, 15], mentioned above, the input application has to be analyzed and specified manually in terms of concurrent task-level model of computation using very specific languages (Stream-C, Handle-C, TDF). This is very time consuming and error prone process because the system designer has to do manually the dependence analysis as well as to learn a specific description language. In contrast, our approach relies on a compiler that fully automatically derives KPN specifications form applications described in common languages like Matlab or C.

2. OUR DESIGN FLOW

To illustrate our design approach that is centered around the use of the KPN model of computation, we show how we integrate the tools COMPAAN and LAURA, we have developed, in a system design flow together with other tools in order to map automatically an application onto a target platform architecture. To make the design flow specific, we demonstrate and evaluate our design flow in the context of a case study in which we map an M-JPEG application onto a platform that consists of a microprocessor and an FPGA running in parallel and communicating with each other via shared memory banks. We have organized the paper in the following way. We give a brief description of the M-JPEG application and the target platform in Section 2.1. This is followed by a step-by-step description of our system design flow in Section 2.2. In Section 3, we present some results that we have obtained. Section 4 concludes the paper.

2.1 M-JPEG and the Platform Architecture

The application we consider is a modified Motion JPEG (M-JPEG) encoder. We have chosen this application because it is a real-life application that is not too complicated, but has enough features to illustrate the use and usefulness of our design flow. Like traditional M-JPEG encoders, the modified M-JPEG encoder compresses a sequence of video frames, applying JPEG [16] compression to each frame in the video sequence. M-JPEG is used for motion pictures compression like MPEG [17] but without inter-frame predictive coding. Our modified M-JPEG encoder, which we further refer to as M-JPEG*, operates on video data in 4:2:2 YUV format and can process each incoming video frame with a different set of quantization and Huffman tables, depending on the output bit-rate and the accumulated statistics from previous video frames. The M-JPEG* encoder application is depicted as a block diagram in Figure 1-a).

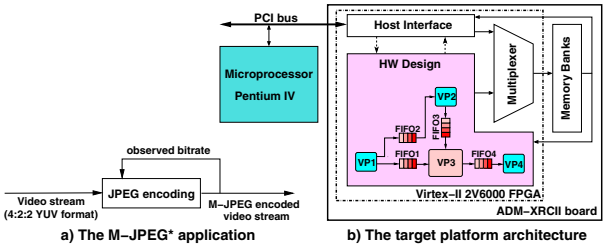

Figure 1: Block diagrams: a) M-JPEG*; b) target platform

We map and run the M-JPEG* application on our target platform architecture which is depicted in Figure 1-b). The platform architecture consists of a microprocessor (i.e., a Pentium IV) running WindowsNT and connected via PCI bus to the ADM-XRCII board manufactured by Alpha Data Parallel Systems, Ltd [18]. The ADM-XRC-

II board is a high performance PCI Card, designed for supporting development of applications using the Xilinx Virtex-II series of FPGAs. The board consists of a Virtex-II 2V6000 FPGA and six ZBT memory banks of size 256k × 32 bit.

2.2 The Mapping

Our system design flow maps an application onto a target platform in a systematic and automated way in a number of steps. We illustrate these steps by mapping the M-JPEG* application onto our platform. Central to our system design flow are the COMPAAN and LAURA tools as shown in Figure 2. The figure shows that an application written in a subset of Matlab is converted to a KPN specification by COMPAAN. Using workload analysis, candidate processes of this specification are selected for mapping on hardware (FPGA). The remaining processes are mapped on the CPU as software. The KPN is written in a particular format in C++ called YAPI [19]. Using a standard C++ compiler, the processes are compiled to run on the CPU on top of a lightweight multi-threading package. The processes that need to be mapped onto the hardware are further processed by COMPAAN to obtain a hierarchical subnetwork. This subnetwork, which is again a KPN, is compiled into hardware using the LAURA tool. Using commercial synthesizers, we obtain the bitstream to map one or more processes onto the FPGA. The communication between the FPGA and the CPU is automatically generated by LAURA.

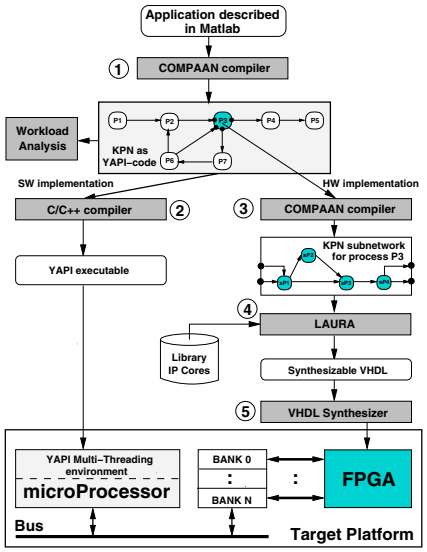

Figure 2: The System Design Flow.

We now look at the various steps in more detail and see how they apply to our M-JPEG* application.

2.2.1 STEP 1

The input of our design flow is an application described in a subset of Matlab. It can be debugged easily and the functional correctness of the application can easily be verified. For the M-JPEG* application, we started from a public domain JPEG codec implementation in C [20]. First, we extracted the encoder part from the implementation and modified it to obtain the M-JPEG* application. Next, we structured our M-JPEG* C-code as a set of routines (functions) that are called by the Matlab code shown in Figure 3.

The Matlab program in Figure 3 is a convenient way to describe the M-JPEG* application. Nonetheless, this program does not reveal the inherent task-level parallelism available in the M-JPEG* due to

```
%parameter NumFrames 8 100;
%parameter VNumBlocks 16 100;
%parameter HNumBlocks 8 100;

for k = 1:1:1,
  [ QTables, HuffTables,
    TablesInfo, EndOfFrame ] = P2_l_DefaultTables();
end

for k = 1:1:NumFrames,

  [ HeaderInfo ] = P1_l_VideoInInit();

  for j = 1:1:VNumBlocks,
    for i = 1:1:HNumBlocks,
      [ Block(j,i) ] = P1_l_VideoInMain();
    end
  end

  for j = 1:1:VNumBlocks,
    for i = 1:1:HNumBlocks,
      [ Block(j,i) ] = DCT( Block(j,i) );

    end
  end

  for j = 1:1:VNumBlocks,
    for i = 1:1:HNumBlocks,

      [ Block(j,i) ] = Q( Block(j,i), QTables );

      [ Packets, StatisticsB ] = VLE( Block(j,i),
                                      EndOfFrame,
                                      HuffTables );

      [ BitRate, StatisticsF,
        EndOfFrame           ] = CtrlF1( StatisticsB );

      [ ] = VideoOut( HeaderInfo, TablesInfo,
                      Packets                 );

    end
  end

  [ QTable , HuffTables,
    TablesInfo          ] = P2_l_CtrlF2( BitRate,
                                         StatisticsF,
                                         QTables,
                                         HuffTables,
                                         TablesInfo  );
end
```

Figure 3: Task-Level specification of the M-JPEG* application in Matlab.

the sequential nature of the program. Therefore, the first step in our system design flow is to convert this sequential program into an executable parallel specification, in our case Kahn Process Network (KPN).

In general, deriving an executable KPN specification by hand for an application is difficult and time consuming. Instead, we relay on the COMPAAN compiler to convert fully automatically the M-JPEG* Matlab program into the KPN specification shown in Figure 4-a). COMPAAN generates a Kahn Process Network as C++ code using the Y–chart Applications Programmers Interface (YAPI) [19]. In YAPI, each process is modeled as a light-weight thread that communicates data with other threads (processes) via unbounded FIFO channels. These channels are accessed using the primitives `read` and `write` to read/write data from/to FIFO channels. The `read` primitive blocks the execution of a process, if the current channel from which a process reads data is empty. The `write` primitive is non-blocking. The blocking-read mechanism accomplishes the inter-process synchronization.

The P1, DCT, Q (Quantizer), VLE (Variable Length Encoding), and VideoOut processes form the central data-flow processing of the M-JPEG encoding algorithm. The CtrlF1 and P2 processes take care of the quantization and Huffman tables adaptation. The CtrlF1 process receives statistics from the VLE process for every incoming block of the current frame that is processed. At the end of the frame, CtrlF1 sends to process P2 a global statistics for the frame as well as the compression bit-rate. Based on the statistics and the bit-rate, P2 computes and sends updated quantization and Huffman tables to process Q and process VLE, respectively.

To obtain a specification that exploits distributed memory, all shared variables in the Matlab code shown in Figure 3 are replaced by

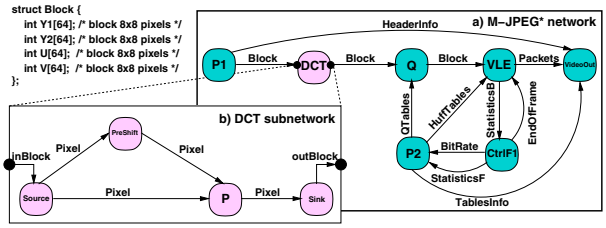

Figure 4: The hierarchical KPN for the M-JPEG* Application.

FIFO channels in the KPN shown in Figure 4-a). For example, the shared variable `Block(j,i)` is distributed over three FIFO channels called Block in Figure 4-a). The type of data communicated over the FIFO channels is the same as the type of the shared variables from which the channels originate. For instance, the type of the variable `Block(j,i)` and the date communicated over the channels with the name Block is the structure shown in the left-up corner of Figure 4.

At the end of step 1 in our design flow, we have obtained an executable specification of the M-JPEG* application as a KPN in YAPI code. When the specification is executed, the YAPI code generates statistics on computational and communicational workload of the application. Based on this information, we perform a manual HW/SW partitioning of the application. We identify the most computational intensive process as a candidate we want to put on hardware to speedup the computation. This is done in step 3 of the system design flow. For the Kahn Process Network shown in Figure 4-a), the most computational intensive process is DCT that performs the Discrete Cosine Transform on every incoming block of pixels. The rest of the processes in the network will be implemented as software and mapped onto the microprocessor.

2.2.2 STEP 2

The processes selected to be put in software, need to execute on the microprocessor of our target platform. For this purpose, we use the YAPI multi-threading environment which is a light-weight multi-threading environment. A standard C/C++ compiler is used to compile the YAPI code of the processes.

All the processes of the M-JPEG* KPN are mapped onto the microprocessor, except for the DCT process which is to be mapped on the FPGA. To integrate the execution of the DCT process on hardware with the software processes on the microprocessor, a small piece of interface code needs to execute on the microprocessor too. This code is given in Figure 5.

```
1   void DCT::main() {
      for (int k=1; k <= NumFrames; k++) {
        for (int j=1; j <= VNumBlocks; j++) {
          for (int i=1; i <= HNumBlocks; i++) {
5           read(inPort, inBlock);
            outBlock = DCT( inBlock );
            write(outPort, outBlock);
          }
        }
10    }
11  }
```

Figure 5: Interface code in YAPI format to connect the Software Processes with the Hardware implementation of DCT.

In line 5 of the code, the YAPI primitive `read()` is used to get data from the input FIFO channel which is connected to the DCT process. The data read from this channel is stored in the variable `inBlock`. The type of this variable is the data structure shown in the left-up corner of Figure 4. This structure consists of four 8×8-pixel blocks. Two blocks for the luminance component (Y1 and Y2) and two for the chrominance component (U and V). Similarly, in line 7, the YAPI primitive `write()` is used to put data that is stored in the variable `outBlock` to the output FIFO channel which is connected to the DCT process. The type of the variable `outBlock` is the same as the type of the variable `inBlock`.

In line 6, the function `DCT()` is called. This function executes a Discrete Cosine Transform (DCT) task implemented as hardware on the FPGA component shown in Figure 1-b). First, the data stored in the input argument `inBlock` of the `DCT()` is uploaded to the memory Bank0 of the *Memory* block shown in Figure 1-b). This is done via the *PCI bus* and the *Host Interface*. Next, the *HW Design* block executes the DCT task and stores the data in the memory Bank1. Finally, this data is downloaded from the memory Bank1 and returned in output argument `outBlock` of function call `DCT()` in line 6.

2.2.3 STEP 3

By performing a workload analysis, we identify a candidate process that is the most computational intensive process of a given KPN. Typically, the code of such process is a nested loop program [10]. We want to implement this process as hardware running on the FPGA in the *HW Design* block of the target platform. For that purpose, we have developed the LAURA tool which generates synthesizable VHDL code from a KPN specification. This VHDL code is suitable for mapping onto FPGAs.

In our case, the candidate process is the DCT process of the KPN in Figure 4-a). Initially, the candidate DCT process is not specified as a KPN. Using again COMPAAN we derive a KPN for this process. This KPN, shown in Figure 4-b), is a hierarchical subnetwork in the M-JPEG*. For this hierarchical subnetwork, we generate synthesizable VHDL. By creating the subnetwork, we exploit more efficiently the parallelism available inside the DCT process. Moreover, we apply automatic type conversion as the hierarchical input/output to/from the DCT subnetwork is data of type Block. Inside the subnetwork, this is converted to streams of pixels (integers). By moving to streams of pixels, we get more fine-grained communication that can be mapped more efficiently onto the FPGA. The type conversion is automatically handled by COMPAAN.

The generation of hardware for the DCT process starts by converting the code in the function call `DCT()` in line 6 of Figure 5, into a Kahn Process Network specification. This is done again by our compiler COMPAAN- step 3 in Figure 2. The code for the `DCT()` function call is described in Matlab as shown in Figure 6.

```
1   for k = 1:1:4,
      for j = 1:1:64,
        [ Pixel(k,j) ] = Source( inBlock );
      end
5   end

    for k = 1:1:4,

      if k <= 2,
10      for j = 1:1:64,
          [ Pixel(k,j) ] = PreShift( Pixel(k,j) );
        end
      end

15    for j = 1:1:64,
        [ Block ] = P_l_PixelsToBlock( Pixel(k,j) );
      end
      [ Block ] = P_l_2D_dct( Block );
      for j = 1:1:64,
20      [ Pixel(k,j) ] = P_l_BlockToPixels( Block );
      end

    end

25  for k = 1:1:4,
      for j = 1:1:64,
        [ outBlock ] = Sink( Pixel(k,j) );
      end
29  end
```

Figure 6: Matlab code of the DCT Process.

The Kahn Process Network (KPN) generated by COMPAAN that corresponds to the Matlab code of the DCT is depicted in Figure 4-b). It shows that this KPN is a subnetwork that implements the DCT process in the M-JPEG* network. The subnetwork consists of four processes. The Source and the Sink processes serve as hierarchical

interfaces to the M-JPEG* network. The Source process transforms the incoming Block data structures to pixels and distributes the pixels corresponding to the luminance components to the PreShift process for preprocessing, while the pixels corresponding to the chrominance components go directly to the P process. The P process executes a 2D-DCT transformation. The Sink process groups the stream of pixels that comes out from the P process in Block data structures.

2.2.4 STEP4

In step 4 of the design flow shown in Figure 2, our tool LAURA transforms the KPN specification generated in step 3 together with predefined IP cores into synthesizable VHDL code. In our example, we provide to LAURA the KPN specification of the DCT. The generation of the VHDL code for this KPN takes place in a number of steps.

First, LAURA creates a platform independent hardware model (HM) for the KPN of the DCT. The obtained hardware model is depicted in Figure 7. It consists of four concurrent virtual processors VP1, VP2, VP3, and VP4 connected in a network that communicate data with each other asynchronously via FIFO buffers. The topology of the HM network is the same as the topology of the input KPN shown in Figure 4-b), as LAURA performs a one-to-one mapping. The virtual processors VP1, VP2, VP3, and VP4 implement the processes Source, PreShift, P, and Sink, respectively.

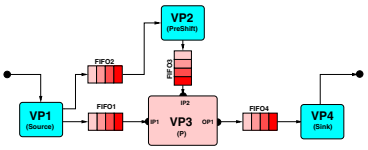

Figure 7: The LAURA hardware model of the DCT subnetwork.

In the second step in LAURA, the hardware model is annotated with additional information about the target platform. This is information about IP cores that are used in the virtual processors, the bit-width of the communicated data, and the type of this data. Also, the size of the hardware FIFO buffers is specified. Furthermore, the notion of a clock event is taken into consideration.

In the last step in LAURA, synthesizable VHDL code is generated that describes the annotated hardware model (HM). We have implemented in LAURA a software procedure called *VHDL Visitor* that generates for each hardware component the correct VHDL syntax. By simply implementing other *Visitor* procedures, we can generate code in other formats, for example, Verilog or SystemC.

2.2.5 STEP5

In the last step of our design flow, we use commercial tools to synthesize and map the VHDL code, generated in step 4, onto the FPGA shown in Figure 1-b). For the synthesis, we used the SinplifyPro tool [21] of Synplicity, Inc. For the placement and routing and for the generation of the configuration file for the FPGA, we used the ISE Foundation package [22] provided by Xilinx.

3. EXPERIMENTS AND RESULTS

In this section, we present some of the results we have obtained by mapping the M-JPEG* application onto the target platform using our system design flow presented in Section 2.

The input to our system design flow was an application described in a subset of Matlab. We started with publicly available sequential C code of a JPEG codec. This code was modified and structured by hand to meet the subset of Matlab that our design flow accepts and to match the features of the M-JPEG* application. The only reason we used Matlab is because COMPAAN uses a simple Matlab parser. Since the model of computation of Matlab and C is the same, you can read "C" every time we speak about Matlab.

The writing of the Matlab code took four days together with the functional testing and debugging. After this preparation work, which is a *one-time* effort only, we started with the mapping of the M-JPEG* application using our system design flow.

Our first experiment was to measure how much time it takes to map the M-JPEG* application onto the target platform using our system design flow. Table 1 shows the processing times for every step in the flow. The last column shows the time needed for every step to finish. The overall time of the whole design flow for the M-JPEG* experiment is around 3 hours and 45 minutes. The column *Manually* indicates that we had to do some manual manipulations. For example, at the end of STEP 1, we had to do a manual HW/SW partitioning that took 30 minutes. In STEP 4, we had to download from Internet, modify and add some IP cores to our library of components, which took 3 hours. However, once the IP cores are in the library any manual manipulations in STEP 4 will disappear.

Table 1: Processing Times (hh:mm:ss).

	COMPAAN	LAURA	*Other tools*	*Manually*	*Total*
STEP 1	00:00:22	–	–	00:30:00	00:30:22
STEP 2	–	–	00:00:35	–	00:00:35
STEP 3	00:00:08	–	–	–	00:00:08
STEP 4	–	00:00:07	–	03:00:00	03:00:07
STEP 5	–	–	00:13:10	–	00:13:10
Overall	00:00:30	00:00:07	00:13:45	03:30:00	03:44:22

The results show that the mapping of the M-JPEG* application onto the target platform is done in a short amount of time - a few hours. The main reason is the great time performance of our tools COMPAAN and LAURA. COMPAAN derives fully automatically a KPN for the M-JPEG in less than a minute after the 4-day preparation work described above. For comparison, a KPN for the same M-JPEG encoder was derived by hand in [23] that took four weeks. LAURA converts fully automatically the KPN of the DCT process in synthesizable VHDL code in a few seconds. For comparison, a hand-made design of this KPN in VHDL will take several days.

In the second experiment, we use our design flow to evaluate the performance of the Kahn Process Network of the DCT mapped onto the FPGA. We measured the time needed for the FPGA implementation to process a single datum of type Block. It took 35 micro seconds at clock frequency of 40MHz. In contrast, the execution of the DCT process running as a program on the microprocessor (clock frequency 1.2GHz) took 98 micro seconds. We conclude, that we got a speedup of 2.8 on the FPGA. We can even improve the speedup by using high-level transformations of the MATTRANSFORM [24] toolbox in STEP 3 (Figure 2). By performing unfolding (unrolling) and skewing (re-timing) transformations, we expect to obtain a speedup of up to 10.

The mapping of the KPN of the DCT is efficient in terms of resource usage. Table 2 shows the FPGA resource utilization. The numbers in the table show that on average only 7% of the FPGA resources are used – 6% is taken by the IP cores and only 1% is taken by the FIFOs and the distributed control generated by LAURA to integrate these IP cores in the KPN of the DCT process. This suggests

Table 2: DCT KPN on VirtexII 2V6000: Device utilization

FPGA resource	*Utilization*	*%*
Number of MULT18X18s	8 out of 144	5%
Number of RAMB16s	4 out of 144	2%
Number of SLICEs	2367 out of 33792	7%
Number of BUFGMUXs	2 out of 16	12%

that we can map on the FPGA not only the DCT process but also other processes of the M-JPEG* network shown in Figure 4-a).

In the final experiment we measured the performance of the complete system: the M-JPEG* Kahn Process Network running on the target platform. We looked at the *throughput* of this system, measured in *frames per second*. For frames in CIF format of 128×128 pixels, the throughput of the system is 10.5 frames per second. This is below the standard minimum real-time throughput of 25 CIF frames per second. We found that the problem is not in the output of our design flow, but in the slow communication of data between the microprocessor and the FPGA. The bottleneck is the 32-bit width PCI bus operating at 33MHz. By switching the PCI bus to 64-bits at a frequency of 66MHz, we can increase the communication speed approximately 4 times. As a consequence, the system should be able to process 25 frames per second in CIF format of 128×128 pixels.

4. CONCLUSIONS

This paper presents a system design flow in which an application written in a subset of Matlab is mapped onto a target platform composed of a CPU and an FPGA in a systematic and automated way. The novelty in this flow is that the CPU and the FPGA run concurrently, thereby exploiting efficiently task-level parallelism. Central to the flow is the use of the Kahn Process Network model of computation. This model inherently expresses applications in terms of distributed control and memory. This is required to get an efficient mapping onto the CPU and the FPGA. In realizing the flow, we have developed and used the COMPAAN and LAURA tools. These tools allow us to quickly go from an application specification in Matlab to an implementation of the application running on the target platform. The hardware mapping shows that LAURA is capable of generating efficient implementations of KPNs. In conclusion, the use of COMPAAN and LAURA (that are still subject to further research) together with other tools results in an efficient design flow for systems that execute high-performance real-time signal processing and multimedia applications.

In this paper, we have demonstrated our system design flow by mapping the M-JPEG* application onto a platform that consists of a CPU and an FPGA. However, our flow is general enough to be used for a systematic mapping of applications onto multiple CPUs and FPGAs. The main reason for this is the Kahn Process Network (KPN) model of computation used in our flow. As the control and memory are distributed in a KPN, no global scheduler is needed. Hence, partitioning a KPN over a number of CPUs and FPGAs can easily be done.

Although we used in our system design flow a standard C++ compiler, a simple research multi-threading environment, and a simple target platform, the obtained results are already promising. Even better results should be achievable when employing, for example, more optimized and robust commercial solutions.

5. ACKNOWLEDGMENTS

This research is partly supported by the PROGRESS program of the Dutch Technology Foundation STW. Also, we would like to thank Vladimir Zivkovic for his contribution in rewriting and structuring the original JPEG C-code.

6. REFERENCES

[1] Gilles Kahn, "The Semantics of a Simple Language for Parallel Programming," in *Proc. of the IFIP Congress 74*. 1974, North-Holland Publishing Co.

[2] Edward A. Lee and Thomas M. Parks, "Dataflow process networks," *Proceedings of the IEEE*, vol. 83, no. 5, pp. 773–799, May 1995.

[3] Bart Kienhuis, Edwin Rijpkema, and Ed F. Deprettere, "Compaan: Deriving Process Networks from Matlab for Embedded Signal Processing Architectures," in *Proc. 8th International Workshop on Hardware/Software Codesign (CODES'2000)*, San Diego, CA, USA, May 3-5 2000.

[4] Alexandru Turjan and Bart Kienhuis, "Storage management in process networks using the lexicographically maximal preimage," in *Proceedings of the IEEE 14th Int. Conf. on Application-specific Systems, Architectures and Processors (ASAP'03)*, The Hague, The Netherlands, June 24-26 2003.

[5] Alexandru Turjan, Bart Kienhuis, and Ed Deprettere, "A technique to determine inter-process communication in the polyhedral model," in *In Proceedings of the "10th International Workshop on Compilers for Parallel Computers, (CPC 2003)*, Amsterdam, The Netherlands, January 2003.

[6] Claudiu Zissulescu, Todor Stefanov, Bart Kienhuis, and Ed Deprettere, "LAURA: Leiden Architecture Research and Exploration Tool," in *Proc. 13th Int. Conference on Field Programmable Logic and Applications (FPL'03)*, 2003.

[7] Wayne Wolf, "A Decade of Hardware/Software Codesign," *IEEE Computer*, vol. 36, no. 4, pp. 35 – 43, Apr. 2003.

[8] Y. Li, T. Callahan, E. Dernell, R. Harr, U. Kurkure, and J. Stockwood, "Hardware-Software Co-Design of Embedded Reconfigurable Architectures," in *Proc. 37th Design Automation Conference (DAC'00)*, Los Angeles, CA, June 5-9 2000, pp. 507–512.

[9] Timothy Callahan, John Hauser, and John Wawrzynek, "The Garp Architecture and C Compiler," *IEEE Computer*, pp. 62–69, April 2000.

[10] J. Villarreal, G. Suresh, G. Stitt, F. Vahid, and W. Najjar, "Improving Software Performance with Configurable Logic," *Kluwer Journal on Design Automation of Embedded Systems*, vol. 7, no. 4, pp. 325 – 339, Nov. 2002.

[11] Vinod Kathail, Shail Aditya, Robert Schreiber, and Bob Rau, "PICO: Automatically Designing Custom Computers," *IEEE Computer*, vol. 35, no. 9, Sept. 2002.

[12] M. Gokhale, J. Stone, J. Arnold, and M. Kalinowski, "Stream-Oriented FPGA Computing in the Stream-C High Level Language," in *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'00)*, April 2000.

[13] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, 1985.

[14] Eric Verhulst, "Beyond the Von Neumann Machine: Communication as the driving design paradigm for MP-SoC from software to hardware," in *Networks on Chips*, Axel Jantsch and Hannu Tenhunen, Eds., pp. 217–238. Kluwer Academic Publishers, 2003.

[15] Eylon Gaspi et al., "Stream Computations Organized for Reconfigurable Execution (SCORE)," in *Proc. 10th Int. Conference on Field Programmable Logic and Applications (FPL'00)*, Aug. 28-30 2000.

[16] Vasudev Bhaskaran and Konstantinos Konstantinides, *Image and Video Compression Standards; Algorithms and Architectures*, Kluwer Academic Publishers, 1995.

[17] W.B. Pennebacker, J.L. Mitchel, C.E. Fogg, and D.J. LeGall, *MPEG Video Compression Standard*, Chapman and Hall, 1996.

[18] "http://www.alpha-data.com/adm-xrc-ii.html," Alpha Data Parallel Systems, Ltd.

[19] E.A. de Kock et al., "YAPI: Application modeling for signal processing systems," in *Proc. 37th Design Automation Conference (DAC'2000)*, Los Angeles, CA, June 5-9 2000, pp. 402–405.

[20] "PVRG-JPEG CODEC 1.1," Portable Video Research Group, Stanford University.

[21] "www.synplicity.com/products/synplifypro/index.html," Synplicity, Inc.

[22] "http://www.xilinx.com/ise/design_tools/index.htm," Xilinx, Inc.

[23] Paul Lieverse, Todor Stefanov, Pieter van der Wolf, and Ed Deprettere, "System Level Design with SPADE: an M-JPEG Case Study," in *Proc. Int. Conference on Computer Aided Design (ICCAD'01)*, San Jose CA, USA, Nov. 4-8 2001, pp. 31–38.

[24] Todor Stefanov, Bart Kienhuis, and Ed Deprettere, "Algorithmic Transformation Techniques for Efficient Exploration of Alternative Application Instances," in *Proc. 10th Int. Symposium on Hardware/Software Codesign (CODES'02)*, Estes Park CO, USA, May 6-8 2002, pp. 7–12.