Stereo Vision for Automotive Industry on Connex Architecture

Bogdan Mîţu, Gheorghe M. Ştefan

Abstract

A stereo vision engine implemented on the Connex Architecture is presented. The Connex architecture refers to a many-processor data parallel machine. The resulting real-time ***65nm*** standard cell design ASIC solution for ***50 frames/second*** stereo vision has ***area < 50 mm^2***, ***power < 1Watt***. Because the problem is solved using a programmable system, additional functionality is easy to be added with no additional costs.

Introduction

This paper is related to the stereo vision engine presented in [1], where the *semi-global matching* (SGM) algorithm is used to perform the stereo vision function. The solution we propose is to use a programmable parallel many-processor with Connex Architecture (CA) to perform the data intense computation requested by the SGM algorithm. The first embodiment of CA was used in the HDTV domain for the H.264 decoding and for the frame rate conversion [2], [3], [4]. The efficiency already proved in solving the previous functions qualifies CA as a promising candidate for providing a very efficient programmable solution for the SGM algorithm. Indeed, **motion estimation,** the core of SGM algorithm, is the most computationally intensive part of the algorithms used in both, H.264 codec and frame rate conversion performed for 1080p HDTV.

Connex many-processor is programmed in ***VectorC***, a C language extension for parallel processing [5]. The extension is made by adding new primitive data types and by extending the existing operators to accept the new data types. In the VectorC programming language the conditional statements have become predication statements.

Connex Architecture

The main challenges for the high performance embedded computing are performance per area (price) and performance per power. The mandatory solution for data intense computing application domains is a data-parallel processor with high *GOPS/mm^2* and *GOPS/Watt* (GOPS: **G**iga **O**perations **P**er **S**econd).

The Connex computational structure performing data parallel computation is a fine-grain network of *small & simple* execution units (EU) working as a many-cell machine. It consists in a linear array of *n* EUs (in the current embodiment *n = 1024*). It provides the **spatial dimension** of the array. Each EU is a 16-bit machine, with a *m* 16-bit **local data memory** (*m = 512* in the present embodiment). This memory

allows storage of *m* *n*-component vectors, generating the **temporal dimension** of the array. The processing array works in parallel with an **IO plane**, a 2-dimension shift register used to transfer data between the array and the external memory (see Fig. 1).

The array is controlled by **CONTR** which works as an instruction sequencer for **CONNEX ARRAY**, while the **IO Plane** transfers data under the control of a specialized hardware configured by the same controller. Thus, data processing and data transfer are two independent processes performed in parallel. Data exchange between the processing array and **IO Plane** is performed in one clock cycle (the **IO Plane** control "watches" for a clock cycle when **CONTR** do not issues a data exchange between the **local data memory** and the register file in the EUs, and then performs the transfer between the **local data memory** and **IO Plane**).

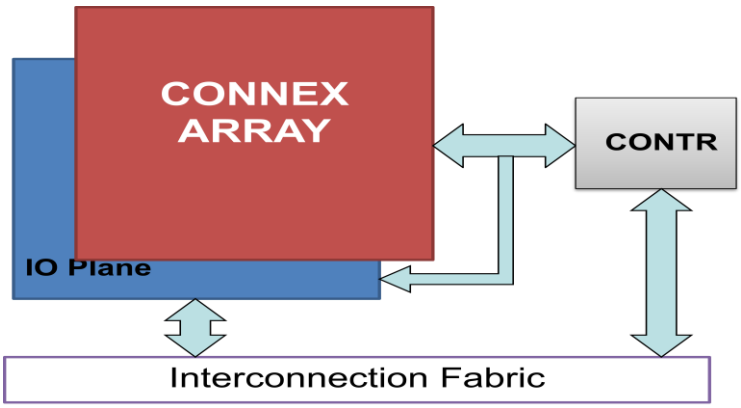

Fig. 1: The Connex System. The processing array is paralleled by the IO Plane which performs data transfers transparent to the processing performed by CONNEX ARRAY.

The core of the system is CONNEX ARRAY, a linearly connected network of EUs and a reduction network (**Reduction Net**) used to perform vector-to-scalar operations (see Fig. 2). Each EU receives from **CONTR** in each clock cycle a new instruction which is executed according to its internal state. The controller receives back data from the EUs through the reduction network. Each EU communicates only with its left and right EU.

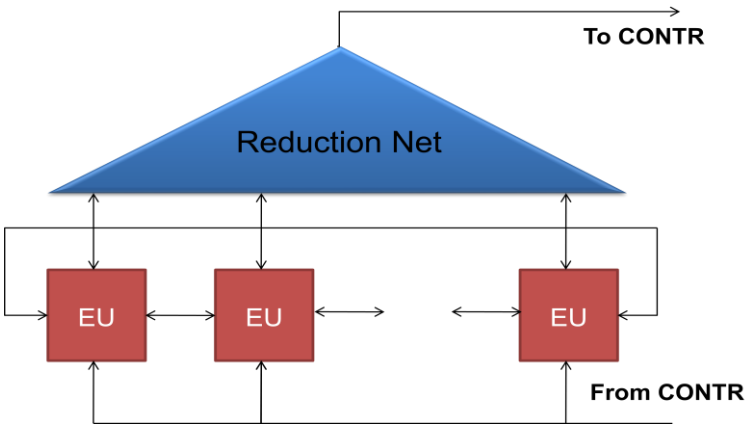

Fig. 2: The Connex Array

The user's view of the data parallel machine is represented in Fig. 3. The linear cellular machine containing *n* EUs performs parallel on randomly accessed vectors. Operations are performed in a constant, small fixed number of cycles. Some generic operations are exemplified in the following:

- **PROCESSING OPERATIONS** performed in the processing array under the control of **CONTR**:
  - **full vector operation**:

    ```
    {carry, v5} = v4 + v3;
    ```

    the corresponding scalar components of the two scalar vectors (**`v4`** and **`v3`**) are added, and the result is stored in the result vector **`v5`** and in the Boolean vector **`carry`**

  - **Boolean operation**:

    ```
    s3 = s3 & s5;
    ```

    the corresponding Boolean components of the two Boolean vectors are ANDed and the result is stored in the result Boolean vector

  - **predicated execution** (see Fig. 4):

    ```
    v1 = s2 ? v3 - v2 : v1;
    ```

    in any positions **where `s2 = 1`** the corresponding components are operated, while in the rest (i.e., **elsewhere**) the content of the result vector remains unchanged (it is a *``spatial if'' statement*)

  - **vector rotate**:

    ```
    v7 = v7 rot n;
    ```

    the scalars of vector **`v7`** is rotated *n* positions right: **`v7[i] = v7[(i+n)mod_n]`**

Fig. 3: The Connex Architectural Space: the content of the local data memories. It is an *m n*-component vectors domain allowing the following type of constant time operations: **`vect_j = vect_i OP vect_k`**

- **INPUT-OUTPUT OPERATIONS** performed in IO Plan:
  - **strided load**:

    `load v5 address burst stride;`

    the content of **`v5`** is loaded with data from the external memory accessed starting from the address: **`address`**, using the bursts **`burst`**, and the stride **`stride.`**

    Example:

    `load v5 5000 128 458;`

    means to load the vector **`v5[0:127]`** (the first 128 bytes of the vector **`v5`**), distributed in the local data memories inside each EU, with **`128`** bytes read starting from the address **`5000`** in the external memory, then to continue loading with others **`128`** bytes read starting from the address **`5458`** in the external memory, then to continue loading with others **`128`** bytes read starting from the address **`5918`** in the external memory, and so on. If

    `burst = stride,`

    then the vector in the external memory is stored in a contiguous space.

  - **gathered load**:

    `gath v3 burst v9 stride;`

    which is a sort of *indirect* load using the vector **`v9`** to generate the request addresses for loading in **`v3`** data from the external memory.

    Example:

    `gath v3 64 v9 32;`

    means to load the vector **`v3`** with **`64`** bytes read starting from the address **`v9[0:3]`** (the first 4 bytes of the vector **`v9`**) in the external memory, then to continue loading with others **`64`** bytes read starting from the address **`v9[0+32:3+32]`** in the external memory, then with others **`64`** bytes read starting from the address **`v9[0+32+32:3+32+32]`** in the external memory, and so on. (Now the stride refers to how the content of the *address vector* **`v9`** is used.)

  - **strided store**:

    `store v7 address burst stride`;

- **scattered store**:

```
scatt v4 burst v13 stride;
```

which is a sort of *indirect* store using the vector **v13** to generate the destination addresses.

Fig. 4: Operating on selected components of the vectors. The operation **vect_j = vect_i OP vect_k** is applied only on the selected components.

In the CONNEX SYSTEM are implemented both, *strided addressing* and *gathered/scattered addressing* because the same data structure must almost all the times transferred in the CONNEX ARRAY in different uniform forms, or in randomly selected shapes. For example, a 2-dimension array of data in some stages of the computations must be loaded in the **Connex Architectural Space** associating to each line a vector or, in other stages of computation associating to each column a vector (the array is rotated). Thus, the strided addressing become very useful, because for the first case the load is performed with **stride = 1**, while for the second case the stride is set equal with the length of the line. The gathered/scattered addressing is used when some "data blocks" are randomly selected to be processed in CONNEX ARRAY.

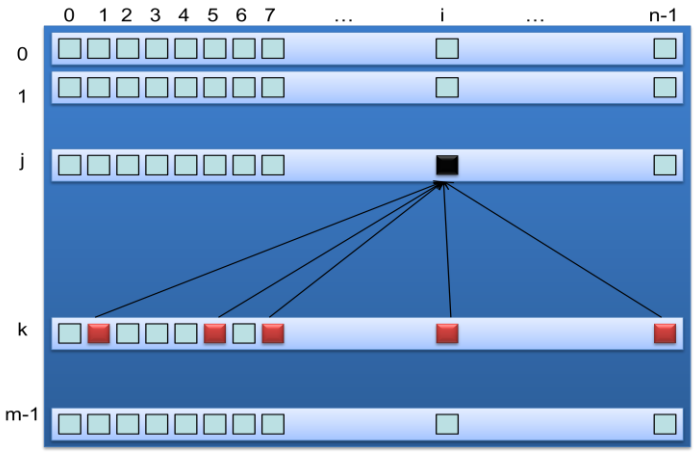

Fig. 5: Log-time reduction operations. For example, all the selected components of the vector *k* are added and stored as one component of the vector *j*.

The reduction network is used to perform vector-to-scalar functions, such as addition, maximum, OR, … . The result is stored in the register file of CONTR or it can be stored back as a scalar component of a vector (see Fig. 5).

The last version of the CA is BA1024B - a SoC for the HDTV market, containing CA1024 (a ConnexArray of 1024 EUs running at 400 MHz) - was produced at TSMC (65 nm standard process) with the following performances:

- 400 GOPS (**G**iga 16-bit **OP**erations per **S**econd)
- >120 GOPS/Watt
- >6.25 GOPS/mm^2.

SGM algorithm on CA

The main components of the SGM algorithm are ZSAD correlation and cost calculation. ZSAD calculation is inherently parallel and appropriate for large-scale SIMD processing. Cost calculation is performed along 8 directions. Along each path operations are sequential but parallel paths can be calculated in parallel. Similar to the FPGA implementation detailed in [1] the 8 paths are computed in 2 passes (see Fig. 4 in [1]). During the 1st pass, accumulated costs for the 4 paths will be stored in external DDR2 memory. In the second pass the costs for the last 4 paths are calculated and combined with the stored costs to make the final decision about pixel disparity.

The rough evaluation of the implementation on Connex Architecture of the algorithm presented in [1] provides the following figures for ***50 frames/sec*** real-time stereo vision:

- number of EUs: $\boldsymbol{n = 340}$ (because the computing is performed on a 640x400 pixels image reduced by a factor of 2 in both dimensions, the biggest dimension of the array involved in computation is 340)
- the clock frequency for Connex Array: $\boldsymbol{f_{connex\_CK} = 200\ MHz}$ (the number of computational steps estimated for 50 frames /sec is less than 200,000,000)
- the external bandwidth: ***1GB/sec*** (because the accumulated costs estimated in the first pass must be stored in external memory and reloaded for the second pass).

Results that the ***SGM algorithm is performed using less than 20% of the computational power of the BA1024B chip*** already designed, implemented, tested and used for writing HDTV applications.

Using the ***Target***[1] design environment the CA (instruction set) can be "tuned" in order to offer for this specific application domain (stereo vision and related computations) additional computational power, optimizing in the same time both, the area of the chip and the power use.

[1] http://www.retarget.com/

The Connex solution for SGM

The proposed solution for running the previously described algorithm is the organization presented in Fig. 6, where the CONNEX SYSTEM is dimensioned for $n = 352$ (organized as 32x11 EUs) and $m = 256$. SRAM BUFFERS are used for fast and flexible data exchange between the CONNEX SYSTEM's internal data memory and the external memory accessed using a p-bit DDR2 CONTR.

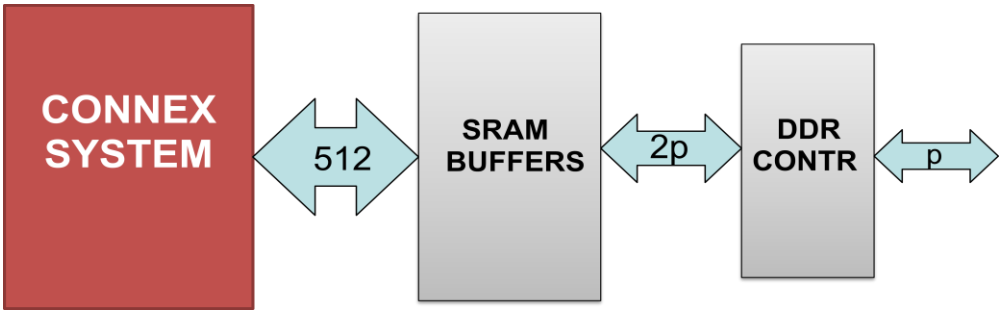

Fig. 6: The Connex Engine for SGM algorithm

The proposed solution can be organized in four stages: (1) evaluation (2) prototype on FPGA, (3) small production on eASIC technology, (4) mass production on ASICs.

Stage 1: Using the existing tools and board built for the frame rate conversion algorithm the SGM algorithm is run on the last version of the CONNEX CHIP in order to evaluate the actual performances of this technology.

Stage 2: The prototype can be developed using *Virtex-6 FPGA Embedded Kit*[2]. In order to implement the basic algorithm, described in [1], the clock frequency for CONNEX SYSTEM will be no more than $f_{connex\_CK} = 200\ MHz$, while the clock frequency of DDR2 CONTR will be $f_{DDR2\_CK} = 250$ MHz for $p=16$.

The CA can be easy adapted to execute **specific instructions** for accelerating the real-time SGM algorithm. We already identified specific instructions to be added to the EUs instruction set which allows additional 20% performance improvement in execution. The saved computational resources can be used to improve the accuracy of the algorithm or to approach new, more detailed aspects of the stereo vision.

Stage 3: The technology offered by ***eASIC***[3] can be used to provide a solution 2 times faster, consuming 10 times less power than the FPGA solution. Because the price of the development and the price of the resulting chip are estimated to be also significantly reduced compared to the FPGA, the small production can be considered in this stage.

Stage 4: For mass production, the ASIC *65nm* implementation of the resulting system is estimated (based on the already implemented CONNEX CHIPS) to have ***area < 50 mm²***, ***power < 1Watt*** for the clock frequencies above defined. But, take note of the additional computational resources resulting from the possibility to drive easy the resulting CONNEX SYSTEM & DDR2 interface up to 500 MHz.

[2] http://www.xilinx.com/products/devkits/EK-V6-EMBD-G.htm
[3] http://www.easic.com/

Because the proposed solution is a programmable one, additional functionality is easy to be integrated on the same system (for example, *full resolution* computation on critical surfaces of the frames).

A specific research project will be able to identify additional ways to adapt the CONNEX approach to the whole project regarding an intelligent driver assistant.

References

[1] Stefan K. Gehring, Felix Eberli, and Thomas Meyer: "A Real-Time Low-Power Stereo Vision Engine Using Semi-Global Matching", in M. Fritz, B. Schiele, and J. H. Piater (Eds.): *Computer Vision Systems*, LNCS 5815, pp. 134-143, 2009.

[2] Gheorghe Stefan, Anand Sheel, Bogdan Mitu, Tom Thomson, Dan Tomescu: "The CA1024: A Fully Programmable System-On-Chip for Cost-Effective HDTV Media Processing", in *Hot Chips: A Symposium on High Performance Chips*, Memorial Auditorium, Stanford University, August, 2006.

[3] Mihaela Malita, Gheorghe Stefan, Dominique Thiebaut: "Not Multi-, but Many-Core: Designing Integral Parallel Architectures for Embedded Computation", in *ACM SIGARCH Computer Architecture News*, Volume 35 , Issue 5, Dec. 2007, Special issue: *ALPS '07 - Advanced low power systems*; communication at *International Workshop on Advanced Low Power Systems* held in conjunction with 21st International Conference on Supercomputing June 17, 2007 Seattle, WA, USA.

[4] Gheorghe Stefan: "One-Chip TeraArchitecture", in *Proceedings of the 8th Applications and Principles of Information Science Conference*, Okinawa, Japan on 11-12 January 2009.

[5] Bogdan Mîţu: "C Language Extension for Parallel Processing", research report, *BrightScale*, 2008. (click here to find it: http://arh.pub.ro/gstefan/Vector\%20C.ppt)