Rapid and Accurate Energy Estimation of Vector Processing in VLIW ASIPs

Erkan Diken, Rosilde Corvino, Lech Jóźwiak
Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands
Email: {e.diken, r.corvino, l.jozwiak}@tue.nl

***Abstract*—Many modern applications in important application domains, as communication, image and video processing, multimedia, etc. involve much data-level parallelism (DLP). Therefore, adequate exploitation of DLP is highly relevant. This paper focuses on effective and efficient exploitation of DLP for the synthesis of vector VLIW ASIP processors. We propose analytical energy models in order to rapidly estimate the energy consumption of a nested loop executed on a VLIW ASIP with respect to different vector widths. The models perform a rapid and relatively accurate energy consumption estimation through combining the relevant information on the application and implementation technology. The analytical energy models are experimentally validated and the validation results are discussed.**

I. Introduction

Computing platforms embedded in various modern devices and processing data intensive communication, multimedia, image processing or signal processing applications are often required to satisfy high performance demands. Moreover, when used in portable equipment, the embedded system must also ensure a low energy consumption, due to the limited battery life. The low energy consumption and high performance are often achieved through highly specialized hardwired processors realized as application specific integrated circuits (ASICs), which can be very efficient, but offers a very-low level of programmability and flexibility. In contrast, application specific instruction-set processors (ASIPs) are programmable and, due to their customization to a specific application, deliver high performance and energy efficiency. Therefore, they are becoming a successful alternative to hardwired circuits.

The computing efficiency provided by ASIPs is boosted by the exploitation of the intrinsic DLP of the application realized through vector/SIMD hardware units. Vector processing is one of the main enablers of computing efficiency due to its regular structure, and low control and interconnect overhead, as studied in [1], [2] and [3]. Moreover, it leads to a narrow program memory and to low number of execution cycles spent in the instruction decoding. On the other hand, having vector units in the ASIP hardware limits flexibility. Meaning that it is efficient and effective only when the vector width of the hardware matches the natural DLP of the application. All other cases result in either less efficiency or effectiveness.

The vectorization effects depend on both the amount of intrinsic DLP of a given application and on the DLP-related configurations of hardware units. The two basic configuration parameters of hardware units related to DLP are their function and their vector width. First of all, the operation types implemented in a vector processing unit have to match the operation functionality of the application. Secondly, the vector width of the hardware units must match the amount of the application's intrinsic DLP. Specifically, the following two cases can be considered regarding the vector width:

- $vector\ width < DLP$: It means that the width of the vector unit is not sufficient to concurrently process all the independent data, this causes more iterations than minimally required, and leads to extra dynamic energy consumption and longer execution time. Moreover, it results in more static energy consumption due to the increased execution time.

- $vector\ width > DLP$: It means that the width of the vector unit is longer than the set of independent data that can be concurrently processed. In this case, there will be an overhead due to a part of the vector unit which is not actually needed. There are some possible optimizations to be considered such as loading of only useful data from memories, disabling the unneeded part of the vector functional units and using banked register file organization to support different vector widths. These optimizations improve the dynamic energy consumption overhead. However, the static (leakage) energy overhead, due to the wider than needed units, remains. Moreover, having applications with different DLPs running on the same hardware unit may require adjusting these optimizations to the need of each application. This may result in an additional overhead.

This paper focuses on effective and efficient exploitation of DLP for the synthesis of vector units of VLIW ASIP processors. We propose analytical energy models in order to rapidly estimate the energy consumption of a nested loop executed on a VLIW ASIP w.r.t. different vector widths. In particular, energy models estimating the energy consumption of a vector computing unit and program memory as a function of vector width are proposed and discussed. Subsequently, validation of the analytical models through experimental research involving the actual mapping of the application onto the processor is presented.

The paper is structured as follows. Sections II and III present the target architecture models and fine-grained application analysis, respectively. Section IV explains the analytical energy consumption models in detail. The following Section V experimentally validates the analytical model and discusses the validation results. Finally, Section VI concludes the paper and lists some future works.

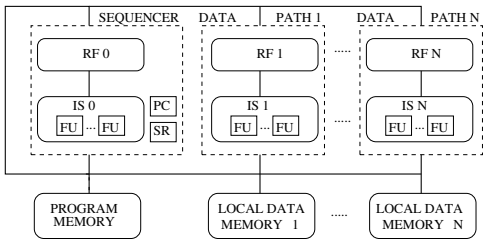

Fig. 1: Generic ASIP architecture template

II. Target Architecture Model

The target ASIP architecture is a VLIW machine capable of executing a parallel software with a single thread of control. Figure 1 depicts a simplified view of the corresponding generic ASIP architecture template. It includes a VLIW datapath controlled by a sequencer that uses status and control registers, and executes a program stored in a local program memory. The data path contains functional units organized in several parallel scalar and/or vector issue slots connected via a programmable interconnect network to register files (RF). The functional units perform computation operations on intermediate data stored in the register files, and only functional units in different issue slots can be triggered and execute parts of an application in parallel. Local memories, collaborating with particular issue slots, enable scalar access for the scalar slots and vector or block access for the vector slots. Both single-instruction multiple-data (SIMD) and multiple-instruction multiple-data (MIMD) processing can be realized on such a processor.

III. Application Analysis

Application analysis is carried out in order to collect the relevant fine-grained data on the application. Analysis is based on the execution traces of the application. Therefore, the application analysis tool consists of the trace generation and trace processing parts.

1) Trace Generation: The tool-set presented in [4] is used to generate a trace. It is based on LLVM [5] compiler infrastructure. Front-end LLVM compiler (clang) is used to compile C code into LLVM intermediate representation (LLVM-IR). Then an instrumentation library is used to generate the instrumented LLVM-IR, and the instrumented LLVM-IR is compiled to the native executable code. Finally, the executable is prompted with a sequence of input stimuli in order to generate the trace. The trace includes run-time instances of each static instruction in LLVM-IR. The trace also includes important information that is not available at the compile time, such as memory addresses for loads/stores, procedure calls, etc. From the execution trace, the corresponding dynamic data dependency graph (DDG) is constructed. A graph node is created for each dynamic instruction instance and an edge is created between each pair of dependent nodes. The DDG only takes flow dependencies between instructions into account, it does not account for anti, output and control dependencies.

2) Trace Processing: The generated trace is processed and analyzed in order to reveal the relevant information on the application. Two analysis tools proposed and implemented are *DLP Estimator* and *Operation Analyzer*.

DLP Estimator: Performs the computation of the parameters $maxDLP^i$ and ρ^i, that are used in the analytical energy models in Section IV. The $maxDLP^i$ corresponds to the maximum available data level parallelism of each instruction. The ρ^i corresponds to the total number of independent occurrences of each instruction.

```
void fir(int vector[BOUND_OL][BOUND_IL]){
  int i, j;
  for(i=0 ; i<BOUND_OL; i++)
  {
    for(j=1; j<BOUND_IL;j++)
    {
(S)   vector[i][j]=(vector[i][j]+vector[i][j-1]) / 2;
    }
  }
}
```

Listing 1: Fir Filter Kernel

The estimations of these parameters is based on the Algorithm 1 presented in [4]. The algorithm applies topological sort traversal to DDG and assigns a time-stamp to each node. This is done separately for all dynamic instances of each instruction. The total number of dynamic instances which has the same time-stamp is defined as $maxDLP^i$. Figure 2 shows the data dependencies of the statement (S) in the code of Listing 1. As it can be observed in the code, there is a data dependence carried by j index ($vector[i][j] = ...vector[i][j-1]$) with dependence distance 1. Consequently, $maxDLP^i$ is limited by the $BOUND\_OL$ variable. The ρ^i parameter, total number of independent occurrences of the statement, is equal to $(BOUND\_IL - 1) * BOUND\_OL$. For keeping the explanation simple, we assume that S corresponds to one instruction.

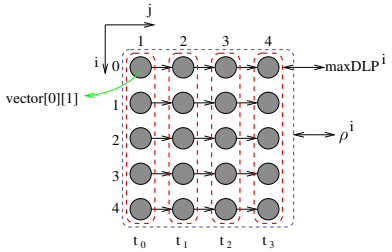

Fig. 2: Data dependencies of fir filter kernel (BOUND_IL=5 and BOUND_OL=5)

Operation Analyzer: Operation complexity analysis is carried out on the traces in order to collect the operation related data on the application. The analysis reveals the kinds of operations and histogram of their occurrences. It also provides information on the computation or memory related operations. Moreover, it reveals information on the data types used in the application. The information can be generated for the whole application, or separately, for each application part.

IV. Analytical Energy Consumption Models

A. The Computing Unit Energy Consumption vs. Vector Width

An analytical model is proposed in order to rapidly estimate the energy consumption of a nested loop executed on a VLIW ASIP w.r.t. different vector widths. The model enables a

rapid and relatively accurate energy consumption estimation by combining the relevant information on the application and implementation technology. It takes both the dynamic and static energy consumption into consideration. The following text introduces and explains the analytical model in detail.

A nested loop body may consist of several statements and each statement may correspond to one or more instructions. Our analysis works on an intermediate representation of the input C code. Therefore, the statements that include more than one instruction are transformed into a sequence of several statements, each representing a simple instruction. A simple instruction corresponds to three address-code format, an instruction with two inputs and one output. First, our model estimates the energy consumption of each instruction in the nested loop. Then computes the total energy consumption by accumulating the value of energy consumption over the occurrences of a given instruction and for all the different instructions.

Energy consumption of an instruction: The total energy consumption of an instruction (E_T^i) is formulated as a function of vector width (w) and is proportional to sum of the dynamic (E_d^i) and static (E_s^i) energy consumption of the instruction (i), as shown in equation (1).

$$E_T^i(w) = E_d^i(w) + E_s^i(w) \tag{1}$$

The dynamic energy consumption of an instruction (E_d^i) depends on the energy spent in data path and sequencer. Energy spent in data path is proportional to the product of the number of iterations (NoI^i) of the same instruction by a normalized energy value (k_d^i) specific to the instruction and vector width (w). k_d^i is a constant and represents the normalized dynamic energy consumption per access to a function unit type that realizes the instruction. Energy spent in sequencer is proportional to the product of the NoI^i by a normalized energy value (k_d^{seq}) specific to the sequencer unit. E_d^i is computed as given in equation (2).

$$E_d^i(w) \propto NoI^i * (k_d^i * w + k_d^{seq}) \tag{2}$$

NoI^i is equal to the result of division of ρ^i of the instruction by dlp^i (equation 3).

$$NoI^i = \frac{\rho^i}{dlp^i} \tag{3}$$

The number dlp^i, DLP amount, that can be processed by hardware is limited by two parameters: one is from the hardware side, namely the vector width, and the other is from the application side, namely the maximum available data level parallelism ($maxDLP^i$) of each instruction. Formulation of dlp^i is given in equation (4), depending on the two possible cases.

$$dlp^i = \begin{cases} w, & w \leq maxDLP^i \\ maxDLP^i, & w > maxDLP^i \end{cases} \tag{4}$$

The static energy consumption of an instruction (E_s^i) is depends on the processor area and program execution time. The product of w, k_s^i and NoI^i is used to represent the static energy consumption of data path. k_s^i is a constant and represents the normalized static energy consumption specific to a given instruction type. NoI^i is used to represent the execution time. Static energy consumption of sequencer is proportional to the product of the NoI^i by a constant (k_s^{seq}) that represents the normalized static energy consumption of the sequencer unit. The corresponding equation is given in (5).

$$E_s^i(w) \propto NoI^i * (k_s^i * w + k_s^{seq}) \tag{5}$$

Energy consumption of a nested loop: The explained model uses data intrinsic to the application, such as $maxDLP$ and ρ, and data related to the implementation technology node such as k_d^i, k_d^{seq}, k_s^{seq} and k_s^i. It also includes the vector width w, which is the explored parameter. Next step is to accumulate the estimates of the instructions in order to get the total energy consumption estimate for the loop. Having N instructions in a loop body, the energy consumption of a loop l is formulated for a width w as shown in equation (6).

$$E_T^l(w) \propto \sum_{i=1}^{N} (E_d^i(w) + E_s^i(w)) \tag{6}$$

Energy consumption of a program: A program may consist of multiple loops. Assuming that the loops are executed on the same data path, the energy consumption (E_T^P) for the whole program is proportional to the sum of the energy consumption of all loops of the program for each w exploration parameter. It is formulated as shown in equation (7) for given $l \in L = \{L_1, ..., L_K\}$.

$$E_T^P(w) \propto \sum_{l=1}^{K} E_T^l(w) \tag{7}$$

B. The program memory energy consumption vs. Vector Width

Energy consumption of the processor's program memory has a significant contribution to the total energy consumption. The energy consumption of the program memory as a function of w is formulated in equation (8). Both the static and dynamic energy consumption of the program memory are taken into account.

$$E_T^{pmem}(w) \propto E_s^{pmem} + E_d^{pmem}(w) \tag{8}$$

Static energy consumption of a program memory (E_s^{pmem}) is dependent on its size (A^{pmem}) and a constant normalized value (k_s^{pmem}) for static energy as formulated in equation (9).

$$E_s^{pmem} \propto k_s^{pmem} * A^{pmem} \tag{9}$$

The program memory size (A^{pmem}) is proportional to word width ($width_{pmem}$), capacity (cap_{pmem}) of the program memory and a normalized constant value (k_{area}^{pmem}) of the program memory area. The corresponding formulation is given in equation (10).

$$A^{pmem} \propto k_{area}^{pmem} * width_{pmem} * cap_{pmem} \tag{10}$$

The word width is largely determined by the complexity of data path, and specifically, the number of operations that can be executed in parallel by the processor. To keep the formula as simple as possible, we formulate the width of the program memory as proportional to the number of parallel elementary data paths (issue slots) (n_{DPs}).

$$width_{pmem} \propto n_{DPs} \tag{11}$$

The program memory capacity is proportional to the sum of latencies of all basic blocks (BB) in the program (equation

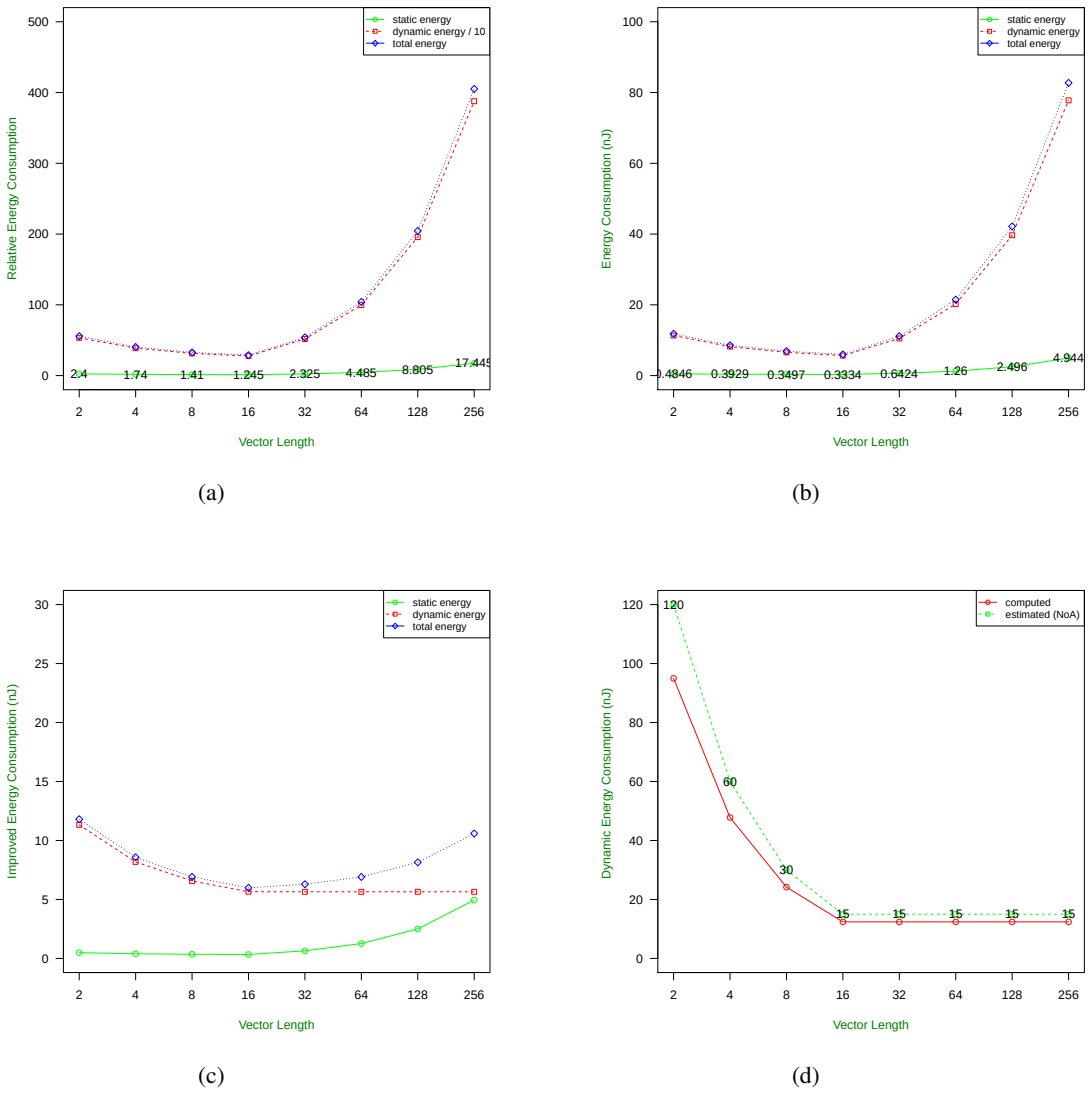

Fig. 3: a) Estimated energy consumption of computing units b) Computed energy consumption of computing units c) Theoretically improved energy consumption of computing units d) Estimated and computed energy consumption of program memory

12). We neglect the instructions that control the program flow, because their influence is negligible for the considered data-dominated applications.

$$cap_{pmem} \propto \sum_{BB=1}^{N} BB_{latency} \tag{12}$$

Latency of a BB is estimated based on the as soon as possible ($ASAP$) scheduling technique. ASAP scheduling estimates the latency of a BB by considering the available number of parallel data paths, as shown in equation (13).

$$BB_{latency} \propto ASAP(BB, DPs) \tag{13}$$

The dynamic energy consumption depends on the number of accesses to the program memory (NoA^{pmem}), word width of the program memory and normalized constant value k_d^{pmem}, as shown in equation (14).

$$E_d^{pmem}(w) \propto NoA^{pmem}(w) * k_d^{pmem} * width_{pmem} \tag{14}$$

Number of accesses to the program memory is formulated in equation (15). It is proportional to the product of a BB latency and the number of iterations of a BB. The sum of the products give the total number of accesses. Since the number of iterations (NoI^{BB}) of a BB equals to the number of iterations of its instructions (NoI^i), NoI^{BB} and NoI^i is used interchangeably.

$$NoA^{pmem}(w) \propto \sum_{BB=1}^{N} (BB_{latency} * NoI^{BB}(w)) \tag{15}$$

V. Analytical Models Validation & Discussion

This section presents validation of the analytical energy consumption models. In order to validate our models, fir filter kernel code in Listing 1 is compiled for and simulated on a processor of the target ASIP technology. The ASIP processor used consists of three issue slots: one vector issue slot to realize vector processing and two scalar issue slots to realize control related computations. Figure 3a refers to estimated relative dynamic and static energy consumption of computing units w.r.t. different vector lengths. Estimation is based on the analytical model demonstrated in subsection IV-A. The estimation uses the analysis data of the code presented in Table I, and normalized static and dynamic energy consumption values for the corresponding operation types.

	$maxDLP$	ρ	operations
Fir Filter	16	240	ADD DIV

TABLE I: Analysis data of fir filter kernel (BOUND_IL = 16 and BOUND_OL = 16)

It is important to stress that the graph presented in Figure 3a does not reflect the exact energy consumption values, but aims at representing the trend of the energy consumption w.r.t. different vector lengths. Figure 3b refers to computed dynamic and static energy consumption of the computing units of the ASIP processor that executes the same code. This computation is based on the similar analytical model, however it uses more accurate data (e.g. number of clock cycles, bit-width, number of accesses) from the simulation of the code on the target architecture. Therefore, the data correspond to exact energy values in nano-joule (nJ). Both the estimated and computed values use the same technology data.

The curve of the graph from estimation is able to well represent the curve of the graph that is based on the simulation results. It clearly shows that the proposed analytical model is appropriate and the captured analysis data are relevant. The results presented in Figure 3a require only one time calculation. On the other hand, the results presented in Figure 3b require several time consuming processor construction, application code compilation and simulation steps. The proposed analytical model and fine-grained application analysis allow for a rapid exploration of vector widths w.r.t. energy consumption without actual compilation and simulation of the application code onto the target processor.

As it can be observed from Figure 3b, the total energy consumption reaches its lowest value when the maximum DLP is equal to the vector length which is 16 for the fir filter kernel. The energy consumption increases when the vector length decreases from 16 to 2, due to the increased control overhead. The vector length increase from 16 to 256 also leads to the increase in the total energy consumption. This is mainly due to the overhead caused by the dynamic energy consumption from the unused data positions in the vector issue slots. Since maximum DLP is 16, the rest of the computations done in the 32, 64, etc. wide functional unit is not necessary. The related energy overhead can be eliminated to a high degree by only loading the useful data from the vector memories and disabling the unneeded part of the functional units. Moreover, banked register files can be used to store different data words. Figure 3c shows theoretical values for the this way improved energy consumption of computing unit.

Figure 3d depicts the computed dynamic energy consumption of the program memory and estimated number of accesses ($NoA^{pmem}(w)$) to the program memory. Since the number of the ASIP data paths and the program do not change, the width and capacity of the program memory also do not change. Therefore, the static energy consumption is the same for different vector lengths, and is not shown in the graph. The dynamic energy consumption is proportional to the number of program memory accesses, because the program memory size is fixed. As it can be observed from the Figure 3d, the estimated and computed values follow the same trend.

VI. Conclusion & Future Work

In this paper, analytical energy models are proposed in order to rapidly estimate the energy consumption of a nested loop executed on a VLIW ASIP with respect to different vector widths. The proposed analytical models are validated. It is proven that the models and the fine-grained application analysis allow for a rapid and relatively accurate exploration of vector widths w.r.t. energy consumption, without any need of the actual time-consuming compilation and simulation.

Our future work involves extension of the application analysis and analytical models in order to explore the more complex communication and memory structures of the ASIP architecture with respect to different vector widths.

Acknowledgment

This work was performed as part of the European project ASAM [6] that has been partially funded by ARTEMIS Joint Undertaking, grant no. 100265.

References

[1] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "SODA: A Low-power Architecture For Software Radio," *SIGARCH Comput. Archit. News*, vol. 34, no. 2, pp. 89–101, May 2006. [Online]. Available: http://doi.acm.org/10.1145/1150019.1136494

[2] J. H. Ahn, W. J. Dally, B. Khailany, U. J. Kapasi, and A. Das, "Evaluating the imagine stream architecture," in *Proceedings of the 31st annual international symposium on Computer architecture*, ser. ISCA '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 14–. [Online]. Available: http://dl.acm.org/citation.cfm?id=998680.1006734

[3] K. van Berkel, F. Heinle, P. P. E. Meuwissen, K. Moerman, and M. Weiss, "Vector processing as an enabler for software-defined radio in handheld devices," *EURASIP J. Appl. Signal Process.*, vol. 2005, pp. 2613–2625, Jan. 2005. [Online]. Available: http://dx.doi.org/10.1155/ASP.2005.2613

[4] J. Holewinski, R. Ramamurthi, M. Ravishankan, N. Fauzia, L.-N. Pouchet, A. Rountev, and P. Sadayappan, "Dynamic Trace-Based Analysis of Vectorization Potential of Applications," in *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2012.

[5] C. Lattner and V. Adve, "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation," in *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)*, Palo Alto, California, Mar 2004.

[6] L. Jóźwiak, M. Lindwer, R. Corvino, P. Meloni, L. Micconi, J. Madsen, E. Diken, D. Gangadharan, R. Jordans, S. Pomata, P. Pop, G. Tuveri, and L. Raffo, "ASAM: Automatic Architecture Synthesis and Application Mapping," in *DSD 2012 - 15th Euromicro Conference on Digital System Design*, Cesme, Izmir, Turkey, September 2012, pp. 216–225. [Online]. Available: http://www.univ-valenciennes.fr/congres/dsd2012/