High-Level Optimization of energy consumed by real-time applications embedded into DSP systems.

Sébastien PIGNOLO, E. Martin, E. Senn, N. Julien (Lab LESTER South Bretany France), B. Saget (Matra Bae Dynamics France)
Sebastien.pignolo@iuplo.univ-ubs.fr

Fax number : *+33134881818,*
Phone:+33681672684
Main author: Sébastien PIGNOLO (PhD student), Lab LESTER, France in partnership with Matra British Aerospace Dynamics

Abstract:

This paper deals with Energy Consumption of Real-time applications running on embedded systems. Assumed that a significant part of the power dissipated by systems is due to Input / Output made between all the chips and especially between commonly used Digital Signal Processors and their tiny internal memories and compared to huge external memories, it is worth settling a strategy to reduce their transfers by keeping inside the processor the more interesting data that will yield to minimize the power consumption. Our method is based on the definition of a density criteria of data utilization and also on the modeling of C source application into two graphs, Graph of Data Dependence and Graph of Expression Dependence that will give information about the behavioral of the data, their precise moments of utilization and the best scheduling of instructions so that data reuse is maximal .First results show that by our method it is possible to reduce by 70 % the energy of some applications.

Key words: *Power optimization, dependence graph, cache, Input / Output access, memory management*

Introduction

This article tackles the problem of minimizing the energy consumed by Real-time software embedded on Digital Signal Processor. Assumed that a significant part of the power dissipated by systems is due to Input/Output made between all the chips and especially between DSPs and their associates internal-external memories, it is worth settling a strategy to decrease the number of the transfers of such a system. In addition, DSPs are not shipped with Data Cache because constructors assume that there is no guarantee of locality for the use of data. At same time compilers only work on local optimization (like registers use) that yield to poor results.

We tackle the energy problem by reducing at high level (C source code) the bandwidth (by managing the internal data memory explicitly like a kind of cache) between the DSP and its external memories by keeping inside the processor the more interesting data that will yield to minimize the power consumed by the peripherals (like DMA, EMIF, Serial Controller), the external bus and external memory. Our method analyzes lifetime of Data and expression dependency all over the application to enhance locality and reusing. First results show that by our method it is possible to reduce by 70 % the energy of some applications.

This paper is divided as follow; first part presents the two graphs used to model the application, then a second part define the density criteria to choose further the “best” data to keep inside the internal memory. A third part indicates the strategy to retrieve from graphs and density criteria the best schedule of expressions that give the highest density for a certain set of Structures (big data). At last we present first results of the use of our method on a Hadamard transform.

High-Level Representation of the source code: the Structure and Expressions Dependence Graphs

Unlike the former studies on Power optimization made at low-level [7][9], the originality of our method is based on the high-level modeling, transformations and optimization of the C source of the software into two graphs. One Graph represents the Dependence between Expressions DEG while the other one represents Dependence between Structures DSG (cf fig.1) (we call “Structure” a data of big size that it is worth optimizing the memory mapping like vectors, arrays). The use of a DEG allow us to schedule expressions without changing the functionality of the application. The DSG shows the link existing between Structures.
DEG = {N,A} where N is the set of the nodes N_i so that N_i is an expression handling a Structure, A_i are the arcs linking the nodes holding the name of the Data that lead the dependence.

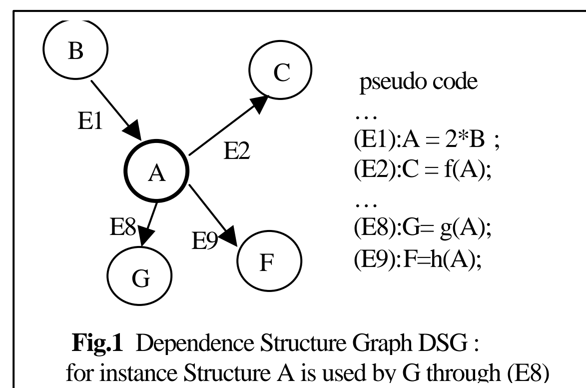

Fig.1 Dependence Structure Graph DSG : for instance Structure A is used by G through (E8)

DSG = {N,A} where N is the set of the nodes N_i so that N_i is a Structure, A_i are the arcs linking the

nodes holding the name of the expression that use it.

Density Criteria

Once a set of Structures is selected, we need a criteria to choose among them those that represent the best benefit to keep inside the internal memory. Density represents how often the elements of a Structure are used in a certain number of expressions (called Observation Window).

density is defined as :

$$\mathbf{D(S_i)}=\frac{number_of_elements_reread_from_Struct_Si}{observation_window}$$

Thus the source code is divided into observation windows of variable size (fig. 2) where groups of Structures are chosen due to the highest density criteria: given the k reread Structures of the application, we have to choose k' (k'≤ k) that contain into the internal memory and that give the highest rate of reuse.

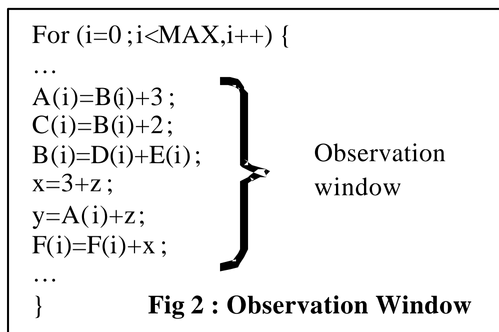

Fig 2 : Observation Window

Then the C source code is re-written with the help of compiler-compiler tool like SUIF [3] to implement a new memory mapping yielding to a brand new source code handling data with respect of our optimizations.

Optimization Strategy

1. Check whether the application is determinist otherwise it is necessary to get information from the programmer or by getting traces from simulations [8][10] to get the results of the conditional branches

2. Get from the C source code the Dependence Graphs of Expressions and Structures

3. Increase the mobility of the expressions by using an anti-dependence skill [9] to remove anti-dependences.

4. PHASE 4 is the stage of optimization, it uses both graphs to find optimal scheduling of expressions and to determine optimal windows that split the code. It uses also criteria like Density to sort Structures that give the best optimization. The size of the window is critical [4] and so influences the quality of the optimization.

5. Modify the C code with SUIF to get a new C source code that implements the mapping of the data in the memories

Optimization PHASE 5

This phase takes both graphs to determine for each Structure the set of expressions that manipulate the Structure. For each Structure it is possible to determine such a window from the DSG and DEG (fig 3).

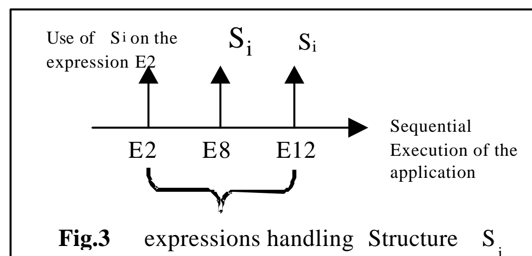

Fig.3 expressions handling Structure S_i

We also determine from DSG and DEG (fig. 5)for every Structure S_i the Extended Window where the Structure S_i is best used. The Extended Window contains all the instructions that manipulate S_i extended by including other expressions needed to compute the set of Structures S_k so that it is worth grouping them in a local execution (fig 4).

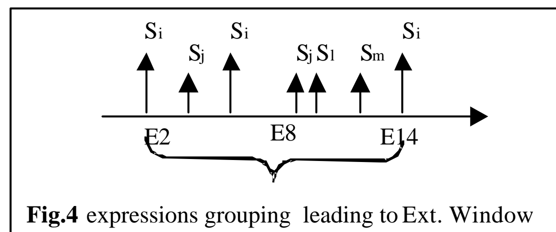

Fig.4 expressions grouping leading to Ext. Window

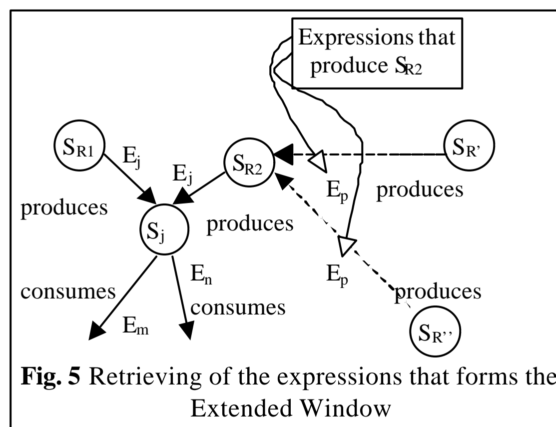

Fig. 5 Retrieving of the expressions that forms the Extended Window

Due to the other "residual" expressions embedded in the Extended Window that do not manipulate S_i we also have to consider the fact where the different uses of S_i are too far from each other, thus there are some lacks in the use of the internal memory (fig 6) that yield to sub-utilization.

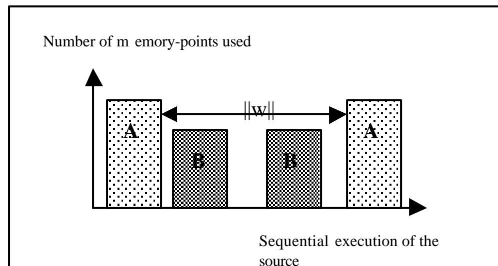

Fig.6 Evaluation of the access-gain with the size of ||w||

To avoid the problem of sub-utilization of internal memory we calculate the benefit to let Structure S_i inside the internal memory instead of flushing the memory place it has taken to replace by new Structures with less density.

Boundary-Effect

Boundary-effect represents how an optimized window can influence the choice of Structures in its neighborhood.

For instance let's say that $ExtW_1$ is the Extended Window of a Structure S_1, S_1 has got its best Density on this optimal Extended Window. S_1 also uses several other Structures S_n. Because of the partial order between all the expressions, S_n has also an optimal Extended Window included, part or all, in the ExtW1 :

$$\{ ExtWS_n \} \subseteq \{ExtWS_1\}$$

The presence of S_n in $ExtW(S_1)$ yield to a potential reuse if we consider that $ExtW(S_n)$ is around of $ExtW(S_1)$ by considering the first use of S_n in the following Observation Window (cf Fig 7) as a reread component (and not only if more than two instances occur) that will increase the density of S_n in F2.

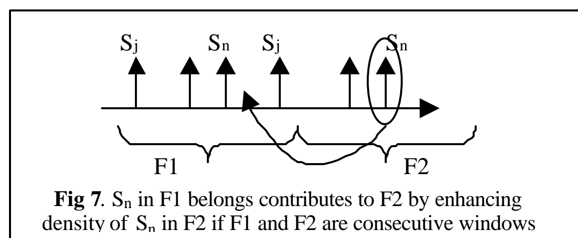

Fig 7. S_n in F1 belongs contributes to F2 by enhancing density of S_n in F2 if F1 and F2 are consecutive windows

So, when calculating the density of a set of Structures in the brand new Window F2 we have to consider the previous uses of all the Structures in the neighborhood.

Global optimization of the application:

The application is split into sub-optimal windows. Each window optimizes a group of Structures for a set of expressions by overlapping Extended-Windows to find the best density on common uses of expressions (fig 8).

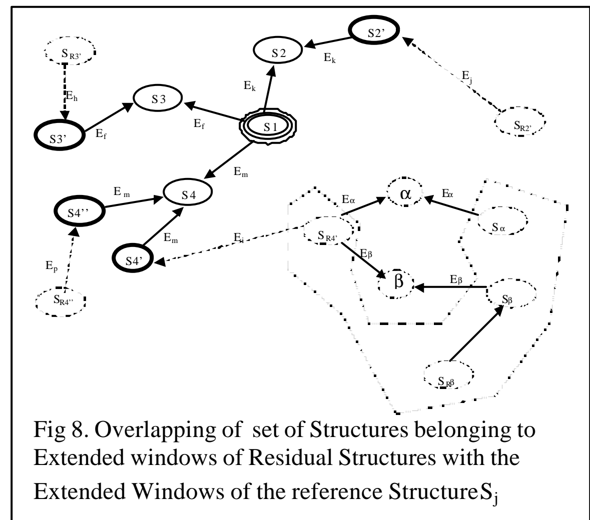

Fig 8. Overlapping of set of Structures belonging to Extended windows of Residual Structures with the Extended Windows of the reference Structure S_j

Once a set of expressions is reordered, we have to consider the other expressions as mobile around the windows already placed. Once an execution window has been optimized, it can't be changed anymore, but it can influence another window not yet optimized (boundary effect). By an glouton-algorithm approach the application is optimized.

First Results

We tried our method on the matrix multiplication for the Hadamard Transform. Our original caching of Structures was tested on a Texas Instrument C6201B DSP evaluation board that we modified to be able to measure with a probe the current that feeds the core processor [6]. We measure on two scenarii the consumption of different strategies.

The Hadamard transformation is given by :

$$[X] := [H].[x][H]$$

⇒ Hadamard matrix H is (**n**,**n**) dimension, it only contains elements **+1** or **-1**
⇒ ***x*** is the image to be transformed
⇒ ***X*** is the transformed image

This transformation only needs additions or subtractions conditioned to the signs of the elements of the Hadamard matrix H.

The Hadamard –Tansform (i) can be decomposed [11] into **X = H.A** :
with X as the result matrix and A is the intermediate matrix **A=x.H**

The product of 2 matrix can be split into 9 sub-products of sub-matrix :

$$\begin{pmatrix} x1 \\ x2 \\ x3 \end{pmatrix} . \begin{pmatrix} H1 H2 H3 \end{pmatrix} = \begin{pmatrix} A11 & A12 & A13 \\ A21 & A22 & A23 \\ A31 & A32 & A33 \end{pmatrix}$$

with x1, x2 et x3 are sub-matrix of dimensions **(n / 3)**x **n** and H1,H2, H3 are sub-matrix of dimensions **n** x **(n / 3)**

We tried our method on matrix of 255 elements (75 elements per x, H matrix). The result matrix A_{ij} is always stored outside the processor to get closed with the philosophy of cache (internal memory is the copy of the highest level of memory).
The schedule of the expressions with our method is indicated below (associated DSG fig 10 and 10bis):

Expressions	Transfers	Internal Memory Content
(E1) :A11= x1.H1	x1,H1, A11	X1, H1 :
(E2):A12= x1.H2	H2, A12	X1, H1*, H2 : data from H1 erased
(E3) :A13= x1.H3	H3, A13	X1, H2*, H3
(E4) :A23= x2.H3	x2, A23	X1*, x2, H3
(E5) :A22= x2.H2	H2, A22	X2, H2, H3*
(E6) :A21= x2.H1	H1, A21	X2, H1, H2*
(E7) :A31= x3.H1	x3, A31	X2*, x3, H1
(E8) :A32= x3.H2	H2, A32	X3, H1*, H2
(E9) :A33= x3.H3	H3, A33	X3, H2*, H3

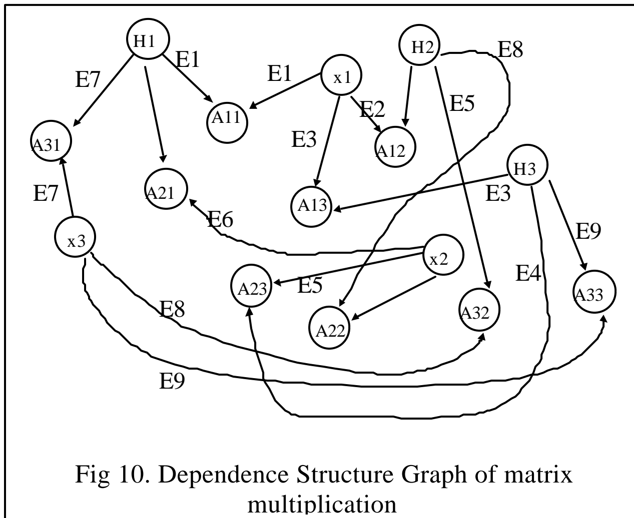

Fig 10. Dependence Structure Graph of matrix multiplication

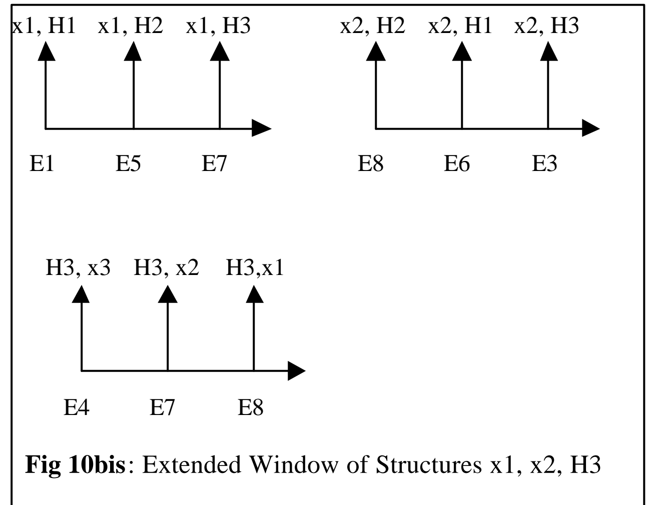

Fig 10bis: Extended Window of Structures x1, x2, H3

<u>Experiment Case 1:</u> All the computations are made in a random order, it may not exist local reuse so we place all the matrix outside the processor, it represents the worst case of computation.
<u>Case 2</u>: Our method is applied, chosen Structures are copied dynamically to enhance reusing. The matrix result is still in the external memory.

Case Version 1	Version 2 (optimized)
Execution Time : T= 5,13 ms	Execution Time : T= 4,89 ms
CPU Core consumption : I_{moy} = 589 mA	CPU Core Consumption : I_{moy} = 630 mA
Total consumption (Joules) : P_{tot}=P(core+E/S+external devices +SDRAM). T	Total consumption (Joules) : V2= 30 % V1

With:

- *SDRAM @ 66,5 MHz and CPU DSP @133MHz*
- *1 SDRAM access consummes 120 mA under 3,3 V*
- *DSP C6x is feeded under 2,5 V*

Tree Clock consumption is 530mA average

With our method, according to the product *Time *Watt* case 2 consumes 70% less power than in case 1 (consumption divided by 4).
V2 solution consumes a little bit more current for the core processor but is faster. External Memory and I/O buffers consume fare more power than the DSP does, so the economy of energy is immediate when Structures reusing is applied.

Conclusion

The more the application is complex (in terms of number of different Structures present in the source code) the more it is possible to optimize the energy. By using a high-level representation of the source-C it is possible to optimize in a very quick way C source without disabling DSP compilers local optimizations [7] [8]. Thus our method is DSP architecture independent. We handle data with explicit addresses so that it is still possible after our optimization to use local classical optimizations. We gave a High Level representation and method that allow opportunities to get a global view of the Structures utilizations so that it is possible to schedule expressions and to choose the best Structures that will yield of energy optimization.

Bibliography

[1] K. Castille, Texas Instrument, doc SPRA486B, Application Report : Preliminary of 06/26/1998 **"TMS320C6201 Power Consumption Summary"**

[2] Carla Schlatter Ellis, Duke university **« The Case for Higher-level power Management »**,

[3] Stanford University Intermediate Format **http://suif.stanford.edu**

[4]Dan Nam Truong, 09/21/1999, Research report IRISA http:www.irisa.fr, **"Software Optimility of locality: the precise inplace of data in memory"**

[5] IMEC, http://www.imec.be "**ATOMIUM project"**

[6] johann Laurent, Eric Martin, Nathalie Julien (lab LESTER, South Bretany), Conf SESAM 2000, Sophia Antipolis Forum on MicroElectronics **"High Level Power Estimation for DSP"**

[7]Bacon, Graham,Sharp,ACM Computing Surveys,Vol. 26, No4, december 1994, **"Compiler Transformations for High Performance Computing "**

[8]Gupta, Miranda, Catthoor, IMEC, **"Analysis of High Level Code Transformation for Programmable Processors"**

[9]Vivek Tiwari, PhD Thesis, Princeton University; Nov 1996, **"Logic and System Design for Low Power Consumption "**

[10] Cheng-Ta Hsieh, Massoud Pedram, IEEE transactions on computer-aided design of integrated circuits and systems, **"Microprocessor Power Estimation Using Profile-driven Program Synthesis "**

[11]E. Martin, J-L Philippe, edition Masson, **"Ingénierie des systems à microprocesseurs, application au traitement du signal et de l'image"**