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

Sébastien PIGNOLO, E. Martin, B. Saget, N. Julien, E. Senn

Lab LESTER, Univ South Bretany France
MATRA BAe Dynamics France

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 data-cacheless Digital Signal Processors shipped with tiny internal scratchpad data-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 yield to minimize the consumption. Our method is based on a data density criteria and on the modeling of the C source application into two Dependence Graphs of Data and Expression that give information about the behavioral of the data, their precise moments of use and the best instruction scheduling 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*

1. Introduction

This article tackles the problem of minimizing the energy consumed by Real-time software embedded on data-cacheless 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 data 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 but with scratch pad internal memory because constructors assume that there is no guarantee of locality in 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 the two graphs and with the help of the density criteria, the best schedule of expressions that give the highest rate of reuse i.e 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.

2. 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.

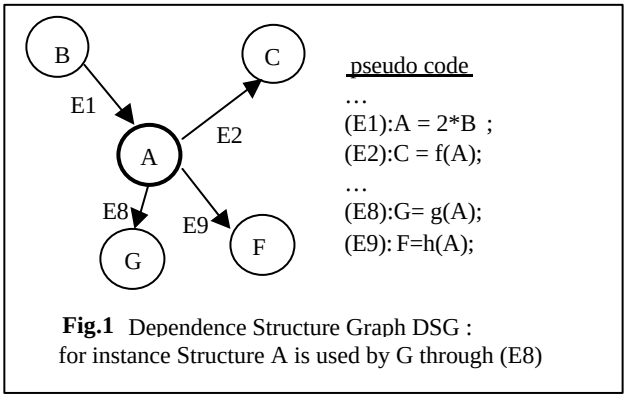

Fig.1 Dependence Structure Graph DSG :
for instance Structure A is used by G through (E8)

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 using it.

3. Density Criteria

Once a set of Structures is selected, we need a criteria to choose among them those that represent the best benefit keeping inside the internal memory.
Density (example fig 2) represents how often the elements of a Structure are used in a certain number of expressions (called Observation Window).

density is defined as :

$$\mathbf{D(S_i)} = \frac{number\_of\_elements\_reread\_from\_Struct\_S_i}{observation\_window}$$

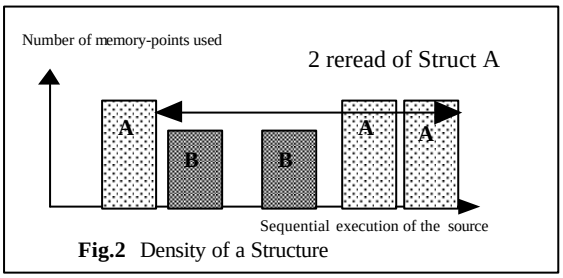

Fig.2 Density of a Structure

$D(S_A)$= 2* Size (A) / {observed window)
$D(S_B)$= 1* Size (B) / {observed window)
here S_A has got the priority to enter in internal memory

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

```
For (i=0 ;i<MAX,i++) {
…
A(i)=B(i)+3 ;
C(i)=B(i)+2;
B(i)=A(i)+E(i);
x=3+z ;
y=A(i)+z ;
F(i)=F(i)+x ;
…
}
```

Observation window

Fig 2-bis : 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.

4. 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 Graph of Expressions

3. Get from the C source code the Dependence Graph of Structures
4. Increase the mobility of the expressions on DGE by using an anti-dependence skill [12] to remove anti-dependences.

5. PHASE 5 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.

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

7. Optimization PHASE 5

This phase takes advantage from both graphs GDE and GDS to determine for every Structure the set of expressions that manipulate the Structure (fig 3).

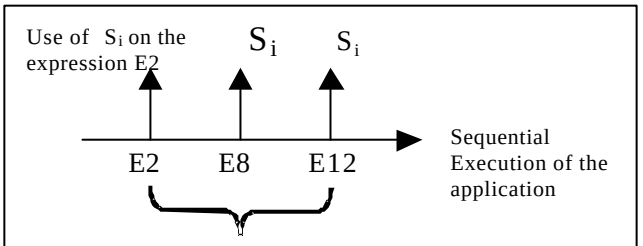

Fig 3. expressions handling Structure S_i

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. Ext. 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).

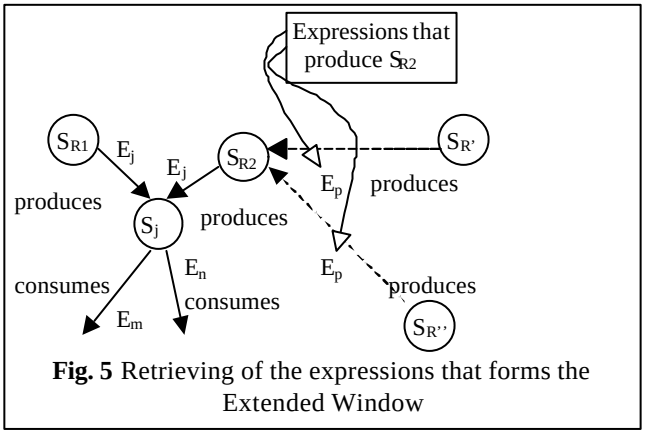

Fig. 5 Retrieving of the expressions that forms the Extended Window

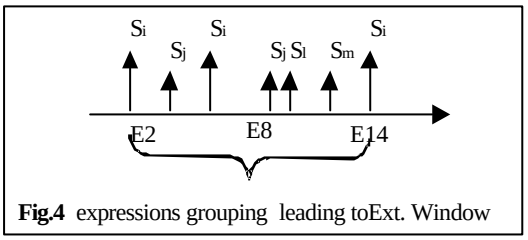

Fig.4 expressions grouping leading toExt. Window

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.

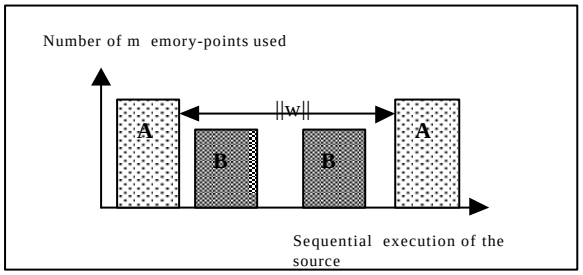

Fig.6 Evaluation of the access-gain with the size of ||w||

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.

8. 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 :

$$\{ ExtWS_n \} \acute{I} \{ExtWS_1\}$$

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.

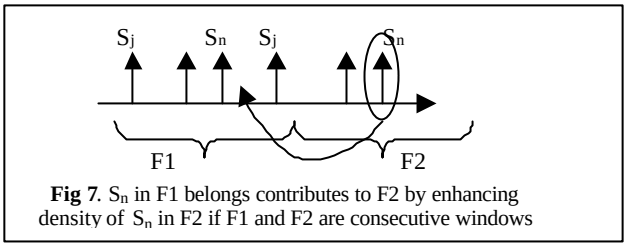

Fig 7. S_n in F1 belongs contributes to F2 by enhancing density of S_n in F2 if F1 and F2 are consecutive windows

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.

9. 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).

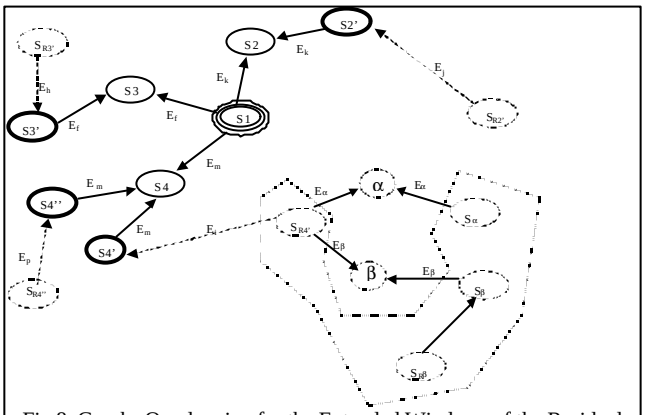

Fig 8. Graphs Overlapping for the Extended Windows of the Residual Structures

Overlapping (Fig 8) with the Extended Windows of the reference Structure S_1 used with S_2, S_3 and S_4. S_4', S_4'' and so on are used to generate S_4, S_3 and S_2. Structures S_R are residual Structures that belong to the Extended Windows for S_4', S_4''and so on.

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). An execution window is complete when the set of chosen Structures has the size of the internal memory because no other Structure can contain on it.
By a glutton-algorithm approach the application is optimized onto successive execution windows.

10.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 scenarios the consumption of different strategies.

The Hadamard transformation is given by :

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

⇒ Hadamard matrix H is (**n**,**n**) dimension, it contains only 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 (t) can be decomposed [11] into **X = H.A** :

with X as the result matrix and A is the intermediate matrix ***A=x.H***

In the case where the matrix H and X cannot fit in the internal memory we rewrite the product of 2 matrix into 9 sub-products of sub-matrix :

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

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), although these matrix contain in the internal memory (internal memory is saturated with two 32ko sub-matrix) these experiments show well the benefit of our method.

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

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).

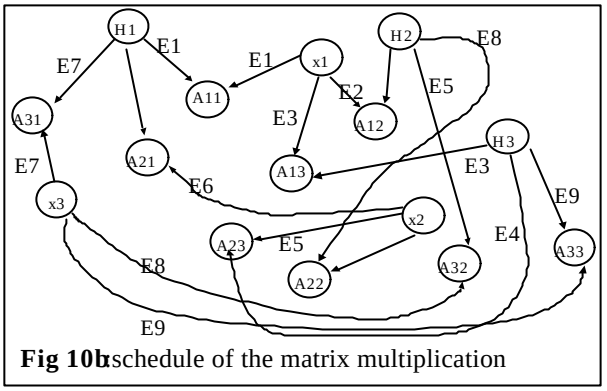

Fig 10b: schedule of the matrix multiplication

Fig 10b shows the schedule resulting of our optimizations and fig 10c indicates the Ext-Window of the Structures onto we performed scheduling, density computation and boundary effect.

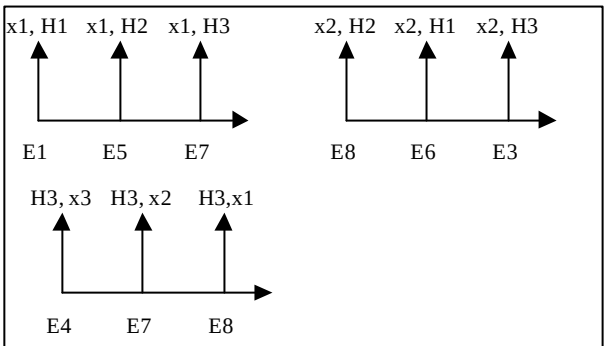

Fig 10c: Ext. Window of Structures x1, x2, H3

Experiment Case 1: 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.

Experiment Case 2: Our method is applied, chosen Structures are copied dynamically to enhance reusing. The matrix result is still in the external memory.

Case Version 1	Version2 (optimized)
Execution Time : T= 5,13 ms	Execution Time : T= 4,89 ms
CPUCore Consumption: I_{moy} = 589 mA	CPUCore Consumption: I_{moy} = 630 mA
Total_Consumption (J) : P_{tot}=P(core+E/S+extern devices +SDRAM). T	Total Consumption (Joules) : V2= 30 % V1

With:

- *SDRAM @ 66,5 MHz*
- *1 SDRAM access consummes 120 mA under 3,3 V*
- *DSP C6x is feeded under 2,5 V @133Mhz*
- *Internal memory is 64ko wide*

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.

11.Conclusion

We are currently trying and experimenting our method on new algorithms like LMS-echo-chancellor algorithm and on a Kalman Filtering. First results show that the more the application is complex (in terms of number of different Structures present in the source code) and handle big-size Structures the more it is possible to optimize the energy. The number of additional source lines and the additional time taken to compute new addresses for memory allocation are compensated by the optimization gain when Structures become bigger.
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 coarse grain optimization to get a global view of the Structures uses, so that it is possible to schedule expressions and to choose the best Structures that will yield to power savings.

Bibliography

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

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

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

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

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

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

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

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

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

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

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

[12]Calland, Darte, Robert, Vivien, "On the removal of anti and output dependences ", Research Report INRIA, Nº2800, February 1996