

# Multidimensional Loop Fusion for Low-Power

Dmytro Lazorenko

*G.E. Pukhov Institute for Modeling in Energy Engineering*

*National Academy of Sciences of Ukraine*

*15 General Naumov Street, Kyiv, 03164, Ukraine*

*E-mail: d.lazorenko@gmail.com*

## Abstract

*Development of semiconductor technology has led to advent of complex digital systems, such as portable, embedded, SoCs, and FPGA devices. Complexity of modern applications and deep-submicron technologies make low-power design attitude compulsory. The higher the level of abstraction of a design that power optimizations are applied, the higher are potential savings. Memory is known to be extremely power consuming. A new technique of loop fusion to optimize a behavioral description of an application before the hardware/software partitioning is presented in this paper.*

## 1. Introduction

Modern portable and embedded systems have become very sophisticated. For example, multimedia applications, portable phones and PDAs. They process large amounts of data by complex algorithms. The speed of battery technology development does not keep up with the increase of energy consumption of new products, hence low-power design solutions are required. Also, low-power results in simpler power distribution, less power and supply bounce, reduction of electromigration and electromagnetic radiation levels [1].

The main source of power consumption in a digital system is dynamic power dissipation. Dynamic power dissipation is the result of switching of the circuit nodes and it can be represented by the following formula:

$$P_{dyn} = C \cdot V_{dd}^2 \cdot \alpha \cdot f \quad (1)$$

where  $C$  is total capacitance,  $V_{dd}$  is power supply,  $f$  is switching frequency and  $\alpha$  is the activity factor (the average fraction of gates switching during one clock period) [1].

Typically dynamic power dissipation is responsible for 80% of total power consumption. Nowadays, the switching activity is largely determined by the system's software [2, 3].

The earlier power optimization decisions are made in the design flow, the higher are potential savings. At system level they can be 50-90%, at behavioral level - 40-70%, at Register-Transfer level - 30-50%, at gate level - 20-30%, at transistor level - 10-20%, and at physical level - 5-10% [4].

Modern applications require a lot of memory, which is known to be extremely power consuming. Memory transfers dissipate a lot of energy. For example, a transfer from an external memory consumes 33 times more than a 16 bits addition. Obviously, the memory size and number of memory transfers should be minimized, so behavioral description of an application must be optimized. The main part of an application code is represented by "for" loops, which are responsible for array processing [5, 6, 7].

## 2. A method of multidimensional loop fusion

Power savings are achieved by minimizing the number of transfers to and from large memories and their size by keeping necessary data in registers or caches. This can be done through loop fusion. A new approach to solving the loop fusion problem is presented in this paper.

Only the main ideas implemented in the algorithm for loop fusion will be presented here. Also, for clarity two-dimensional arrays will be used in examples. Loop fusion is performed via graph transformation. The way to create graph for a c++ code, containing multidimensional loops, is illustrated on Fig. 1 and Fig.2.

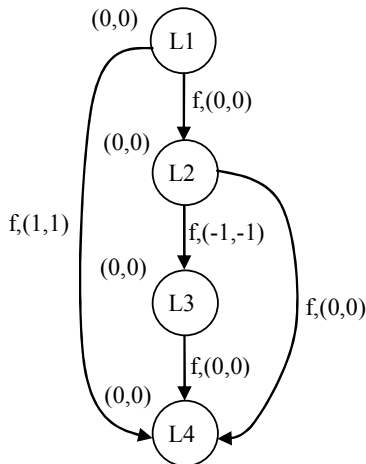
```

...
for (int i = 0; i < N; i++)
for (int j = 0; j < N; j++)
a1[i][j] = f1(i,j);
...
for (int i = 0; i < N; i++)
for (int j = 0; j < N; j++)
a2[i][j] = f2(a1[i][j]);
...
for (int i = 0; i < N - 1; i++)
for (int j = 0; j < N - 1; j++)
a3[i][j] = f3(a2[i+1][j+1]);
...
for (int i = 1; i < N; i++)
for (int j = 1; j < N; j++)
a4[i][j] = f4(a1[i-1][j-1], a2[i][j], a3[i][j]);
...

```

**Fig. 1. An example of C++ code**

Graph corresponding to above C++ code is presented on Fig. 2.

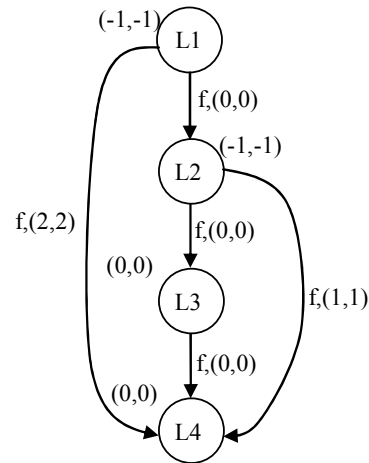


**Fig. 2. Graph corresponding to C++ code above**

Each vertex of the graph corresponds to a certain calculation of an array. For example, vertex L1 corresponds to the calculation of a1 array in the first loop statement. L2 corresponds to the calculation of a2 array in the second loop statement. There exists flow dependence between first and second loops. It is represented in the graph by edge L1→L2. The edge is labeled “f” for flow-dependence. The edge is also assigned a set of weights; the number of weights is equal to the number of array dimensions. Weights are calculated as follows. If element  $a2[i_1, i_2, \dots, i_n]$  is calculated using element  $a1[i_1 - d_1, i_2 - d_2, \dots, i_n - d_n]$ , then weight for k-th iteration variable is calculated as  $i_k - (i_k - d_k) = d_k$ . Hence, the set of weights for the

edge is going to be  $(d_1, d_2, \dots, d_n)$ . Before graph transformation weights of its vertices are going to be  $(0, \dots, 0)$ . Weights of vertices are used after graph transformation for creation of the fused loop code.

Similar to how it was done in [8] for one-dimensional loop fusion, to make fusion of all multidimensional loops possible, it is necessary to make values of all edges weights nonnegative. In case of the example on Fig. 1 and Fig. 2 weights of the edge L2→L3 must be increased by 1 to 0. Then the weight of vertex L2 must be decreased by the same value (1) to  $(-1, -1)$ . At the same time weights of all ribs leaving vertex L2 must be increased by 1 and weights of all ribs coming into L2 must be decreased by 1. The same transformations must be performed on all ribs with negative weights while moving to the first vertex of the graph. Transformed graph is presented on Fig. 3.



**Fig. 3 Transformed graph.**

The code of corresponding fused loop is shown on Fig. 4.

```

...;
for (i = 2; i < N; i++)
for (i = 2; j < N; j++) {
a1[i][j] = f1(i,j);
a2[i][j] = f2(a1[i][j]);
a3[i-1][j-1] = f3(a2[i][j]);
a4[i-1][j-1] =
f4(a1[i-2][j-2], a2[i-1][j-1], a3[i-1][j-1]);
}
...;

```

**Fig. 4 Fused loop code.**

Fusion of loops where output dependence is present is also possible. Example of C++ code with three loops is presented on Fig. 5. The first and the third loops have output dependence.

The graph corresponding to this code example is presented on Fig. 6. The graph is created similarly to

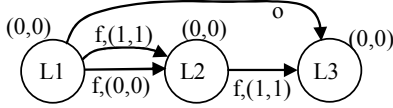
the graph for the code example on Fig. 1. The edge from vertex L1 to vertex L3 is labeled “o” for output dependence. The edge does not have weight.

```

...
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    a[i][j] = f1(i,j);
...
for (int i = 1; i < N; i++)
  for (int j = 1; j < N; j++)
    b[i][j] = f2(a[i][j],a[i-1][j-1]);
...
for (int i = 0; i < N - 1; i++)
  for (int j = 0; j < N - 1; j++)
    a[i+1][j+1] = f3(b[i][j]);
...

```

**Fig. 5. An example of C++ code with output dependence.**



**Fig. 6. Graph corresponding to C++ code above**

To make fusion of loops on Fig. 5 possible, it is necessary to transform the graph on Fig. 6 in the following way.

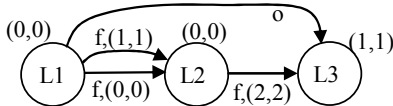
The values of **a** array elements are used for calculation of **b** array elements and then are overwritten in the third loop. Fusion of these loops is possible if calculations are organized in such a way, so that values of **a** array elements are overwritten only after they were used for **b** array elements calculation.

Similar to how it was done in [8] for one-dimensional loop fusion weight **w** of vertex L3 (terminal vertex for o-edge) must meet the following condition:

$$w_i(L3) \geq w_i(L1) + \max \{w_i(f_{edge\ leaving\ L1})\} \quad (2)$$

the *i*-th weight of vertex L3 must be more or equal to the sum of *i*-th weight of vertex L1 and maximum value of *i*-th weight of *f*-edges leaving vertex L1.

Transformed graph and the C++ code corresponding to it are presented on Fig. 7 and Fig. 8 respectively.



**Fig. 7 Transformed graph.**

It is also possible to take into account dependences for calculations inside loops when analyzing them for possibility of fusion. An example of C++ code with two

loops which can not be fused is presented on Fig. 9. Graph corresponding to the code is shown on Fig. 10.

```

...
for (int i = 2; i < N; i++)
  for (int j = 2; j < N; j++)
  {
    a[i][j] = f1(i,j);
    b[i][j] = f2(a[i][j],a[i-1][j-1]);
    a[i-1][j-1] = f3(b[i-2][j-2]);
  }
...

```

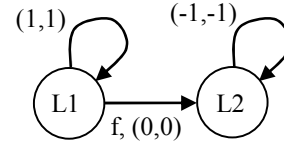
**Fig. 8 Fused loop code.**

```

...
for (int i = 1; i < N; i++)
  for (int j = 1; j < N; j++)
    a[i][j] = f1(a[i-1][j-1]);
...
for (int i = N - 2; i >= 0; i--)
  for (int j = N - 2; j >= 0; j--)
    b[i][j] = f2(b[i+1][j+1], a[i][j]);
...

```

**Fig. 9. An example of C++ code with loops which can not be fused.**



**Fig. 10. Graph corresponding to C++ code above**

Values of array elements  $a[i][j]$  are calculated using elements  $a[i-1][j-1]$ , this is represented in the graph by the loop edge (vertex L1). Also, values of array elements  $b[i][j]$  are calculated using elements  $b[i+1][j+1]$ , this is represented in the graph by the loop edge (vertex L2). These edges are assigned sets of weights. The number of weights in a set is equal to the number of array dimensions. These weights are calculated the following way. If element  $a[i_1, i_2, \dots, i_n]$  is calculated using element  $a[i_1-d_1, i_2-d_2, \dots, i_n-d_n]$ , then weight for *k*-th iteration variable is calculated as  $i_k - (i_k - d_k) = d_k$ . Hence, the set of weights for the edge is going to be  $(d_1, d_2, \dots, d_n)$ .

As it is shown by example above, loops can not be fused if their vertices are connected by an *f*-edge and have loop edges with corresponding weights of the opposite sign.

## 4. Experiment

To verify the effect of loop fusion on power consumption an experiment was conducted. Two simple c-programs were downloaded to a

microcontroller unit (Texas Instruments MSP430F149) and the consumption current was measured with a multimeter. Text of these programs is presented on Fig. 11 and Fig. 12.

```

__root int a1[50];
__root int a2[50];
__root int a3[50];
__root int a4[50];
int main()
{
    int i;
    for(;;)
    {
        for ( i=0; i<50; i++) a1[i] = i;
        for (i = 0; i<50; i++) a2[i] = a1[i] * 2;
        for (i = 0; i<50; i++) a3[i] = a2[i] - 1;
        for (i = 1; i<50; i++) a4[i] = a3[i] * 2 + 1;
    }
}

```

**Fig. 11. Program 1. Original loops.**

```

__root int a4[50];
void main(void)
{
    int i, a1, a2, a3;
    for(;;)
    {
        for (i = 0; i < 50; i++) {
            a1 = i;
            a2 = a1 * 2;
            a3 = a2 - 1;
            a4[i] = a3 * 2 + 1;
        }
    }
}

```

**Fig. 12. Program 2. Fused loop.**

Consumption current for the first program is 680  $\mu\text{A}$ , and for the second – 538  $\mu\text{A}$ . As a result of loop fusion 20.8% gain in consumption current is achieved.

### 3. Conclusions

Experimental data show that loop fusion transformation of the original code of an application results in decrease of power consumption of the application.

The method offered in this paper enables analysis of multidimensional loops for possibility of their fusion. The method produces better result than the one presented in [6], as it allows fusion of all loops. Also, the method allows taking in account calculations performed inside a loop during analysis of loops for the possibility of their fusion; this is not possible with the method from [6].

### 10. References

- [1] H.J.M. Veendrick, Deep-Submicron CMOS ICs. From Basics to ASICs. Kluwer academic publishers, 2000.
- [2] F. Poppen, Low Power Design Guide. OFFIS Research Institute. <http://www.offis.de>
- [3] H.S. Kim, M.J. Irwin, N. Vijaykrishnan, M. Kandemir, Effect of compiler optimizations on memory energy. 2000 IEEE Workshop on Signal Processing Systems, SiPS 2000, pp. 663 - 672, Oct. 2000.
- [4] J. Sproch, High Level Power Analysis and Optimization. Tutorial. In Proc. 1997 International Symposium on Low Power Electronics and Design. 1997.
- [5] A. Fraboulet, G. Huard, and A. Mignotte, Loop Alignment for Memory Accesses Optimization. In Proc. 1999 Twelfth International Symposium on System Synthesis. Proceedings (ISSS'99). IEEE Computer Society Press, pp. 71-77, Nov. 1999.
- [6] A. Fraboulet, K. Kodary, and A. Mignotte, Loop fusion for memory space optimization. In Proc. 14th International Symposium on System Synthesis, 2001, Proceedings, pp. 95 - 100, 2001.
- [7] F. Catthoor, F. Franssen, S. Wuytack, L. Nachtergaele, H. De Man, Global communication and memory optimizing transformations for low power signal processing systems. Workshop on VLSI Signal Processing, VII, pp. 178 - 187, Oct. 1994.
- [8] D.I. Lazorenko, Algorithm for one-dimensional loop fusion in the source code of an application for low-power. Information processing systems, Kharkov Air Force University, Issue 8(66), pp. 2-12, 2007. (in Russian)