Domain-Independent Kernel-Based Intermediate Representation for Automatic Parallelization of Sequential Programs

José M. Andión, Manuel Arenaz, Juan Touriño*,[1],[2]

* *Computer Architecture Group, Department of Electronics and Systems, University of A Coruña, Spain*

ABSTRACT

Parallelizing compilers typically use statement-based standard intermediate representations (e.g., Data Dependence Graph, Control Flow Graph, Dominance Tree) that hinder discovering the parallelism available in sequential programs. This paper presents a new IR based on the domain-independent concept-level kernels (e.g., induction, reduction, recurrence) recognized by the XARK compiler framework. Such kernel-based IR hides the complexity of the implementation details, and exposes multiple levels of parallelism to the compiler.

KEYWORDS: automatic parallelization; domain-independent kernel; XARK compiler framework

1 Introduction

The XARK compiler framework [ATD08] provides a complete, robust and extensible solution for the automatic recognition of domain-independent concept-level computational kernels. XARK was shown to be effective to characterize a significant amount of the regular and irregular computations carried out in full-scale Fortran77 applications [ATD07] from SPEC CPU2000, Perfect benchmarks, Sparskit-II and PLTMG. In order to widen the scope of application of XARK to other programming languages that use pointers, we have developed

[1]E-mail: {jose.manuel.andion,manuel.arenaz,juan}@udc.es

[2]This research was supported by the Ministry of Science and Innovation of Spain and FEDER funds of the European Union (Project TIN2007-67537-C03) and by the FPU Program of the Ministry of Education of Spain (Reference AP2008-01012).

```
 1  void gradient_aprox (long *sum, unsigned char **data, int cols, int Y, int X, int G[3][3])
 2  {
 3    int I, J;
 4    for(I=-1; I<=1; I++)
 5      for(J=-1; J<=1; J++)
 6        (*sum) = (*sum) + (int)((*((*data) + X + I + (Y + J)*cols)) * G[I+1][J+1]);
 7  }
 8
 9  int main(void)
10  {
11    int            originalImage_rows, originalImage_cols, edgeImage_rows, edgeImage_cols;
12    unsigned char* originalImage_data, edgeImage_data;
13    int            X, Y, I, J, GX[3][3], GY[3][3];
14    long           sumX, sumY, SUM;
15
16    for(Y=0; Y<=(originalImage_rows-1); Y++)  {
17      for(X=0; X<=(originalImage_cols-1); X++)  {
18        sumX = 0;
19        sumY = 0;
20
21        if(Y==0 || Y==originalImage_rows-1)
22          SUM = 0;
23        else if(X==0 || X==originalImage_cols-1)
24          SUM = 0;
25        else {
26          gradient_aprox (&sumX, originalImage_data, originalImage_cols, Y, X, GX);
27          gradient_aprox (&sumY, originalImage_data, originalImage_cols, Y, X, GY);
28          SUM = abs(sumX) + abs(sumY);
29        }
30        if(SUM>255) SUM=255;
31        if(SUM<0) SUM=0;
32
33        *(edgeImage_data + X + Y*originalImage_cols) = 255 - (unsigned char)(SUM);
34      }
35    }
36  }
```

Figure 1: Source code of the Sobel application.

a simple and fast algorithm to build the Gated Single Assignment (GSA) form on top of the Static Single Assignment (SSA) form available in modern production compilers [AAT08]. In addition, the design of an interprocedural GSA form to support automatic kernel recognition across procedure boundaries is work in progress. The final goal is to propose a new kernel-based IR [AAT10] that exposes parallelism to the compiler and that provides a unique view of different codifications of the same kernel. In the literature, only approaches based on statement-level standard IRs have been proposed [SP01, Vah02, RVDB09]. The poster will show the differences and commonalities of several implementations of the Sobel edge filter.

2 The Sobel Edge Filter

The Sobel edge filter is a well-known algorithm widely used in image processing and computer vision. This algorithm detects the edges of an image, that is, those pixels whose intensity is very different from the intensity of the neighbor pixels. For each pixel, the algorithm computes the gradient value that provides the largest increase from light to dark. For illustrative purposes, consider the interprocedural implementation shown in Figure 1. For each pixel of the original image (see loops in lines 16–17), the procedure `gradient_aprox` computes a convolution of the 3×3 matrix `GX` and the intensity of the pixel and its eight neighbors (line 26). A similar convolution with the 3×3 matrix `GY` is also computed (line 27).

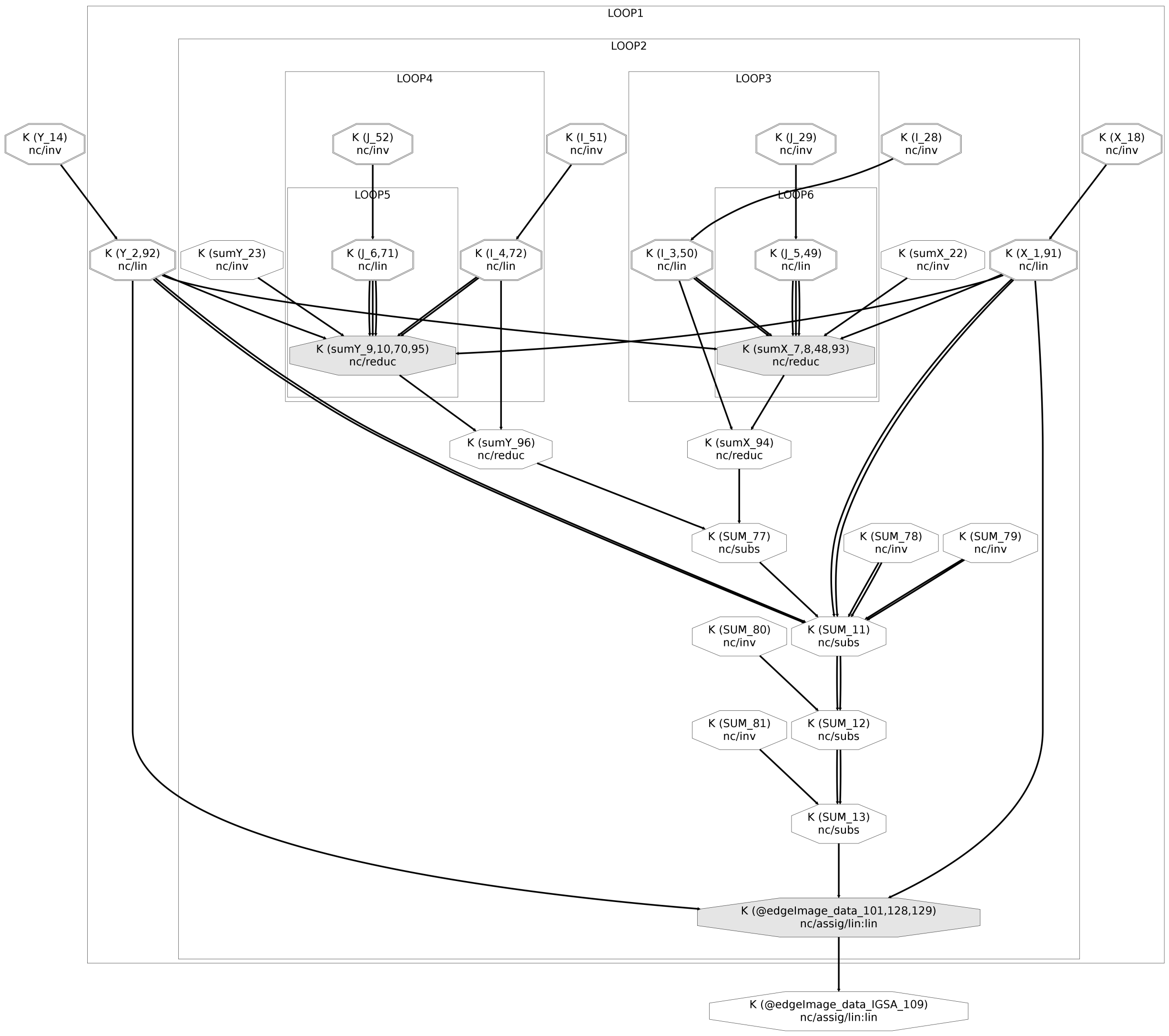

Figure 2: Kernel-based intermediate representation of the Sobel application.

Finally, the sum of the absolute values of the two convolutions is truncated to the interval $[0, 255]$ (lines 30–31) before being stored in the output filtered image (line 33). Note that, in order to compute the convolutions, the image boundaries are not processed (lines 21–24).

3 Kernel-Based Intermediate Representation

Our new kernel-based IR consists of two graphs, a KDDG that shows the kernels and the dependences between them, as well as a KCFG that groups kernels into loop execution scopes. Node labels $K(x_1 \ldots x_n)$ represent a kernel whose output variables are $x_1, \ldots, x_n$ in the GSA form. In addition, nodes are labeled with the type of kernel computed during its execution, namely, *nc/inv* for initialization of variables to constant values (see $K(sumY_{23})$ in LOOP2); *nc/lin* for linear inductions ($K(Y_{2,92})$ in LOOP1); *nc/subs* for unpredictable values at compile-time (e.g., subscripted subscripts, pointer dereferences) (see $K(SUM_{77})$ in LOOP2); *nc/reduc* for scalar reductions ($K(sumY_{9,10,70,95})$ in LOOP5); and *nc/assig/lin:lin* for the initialization of a 2D array variable using a linear access pattern in both dimensions ($K(@edgeImage\_data_{101,128,129})$ in LOOP2).

The kernel-based IR of Figure 2 exposes several kernels that can be parallelized (depicted as shaded nodes): $K(sumX_{7,8,48,93})$ and $K(sumY_{9,10,70,95})$ as parallel scalar reductions, and $K(@edgeImage\_data_{101,128,129})$ as a parallel regular assignment. In our experiments, we have checked that the same parallel kernels are discovered in several implementations that use array/pointers to access image values, procedure calls to compute the gradient values, global variables for error handling, and different control flows to avoid the computations of image boundaries.

4 Conclusions

This paper has shown that our domain-independent kernel-based IR exposes the multiple levels of parallelism available in sequential programs, while providing a unique view of different implementations of the same algorithm. An interprocedural implementation of the Sobel edge filter has been used for illustrative purposes.

References

[AAT08] Manuel Arenaz, Pedro Amoedo, and Juan Touriño. Efficiently Building the Gated Single Assignment Form in Codes with Pointers in Modern Optimizing Compilers. In *Proceedings of 14th International Euro-Par Conference (Euro-Par)*, volume 5168 of *Lecture Notes in Computer Science*, pages 360–369, Las Palmas de Gran Canaria, Spain, 2008.

[AAT10] José M. Andión, Manuel Arenaz, and Juan Touriño. Automatic Partitioning of Sequential Applications Driven by Domain-Independent Kernels. In *Proceedings of 15th Workshop on Compilers for Parallel Computing (CPC)*, Vienna, Austria, 2010.

[ATD07] Manuel Arenaz, Juan Touriño, and Ramón Doallo. Program Behavior Characterization Through Advanced Kernel Recognition. In *Proceedings of 13th International Euro-Par Conference (Euro-Par)*, volume 4641 of *Lecture Notes in Computer Science*, pages 237–247, Rennes, France, 2007.

[ATD08] Manuel Arenaz, Juan Touriño, and Ramón Doallo. XARK: An eXtensible framework for Automatic Recognition of computational Kernels. *ACM Trans. Program. Lang. Syst.*, 30(6), 2008.

[RVDB09] Sean Rul, Hans Vandierendonck, and Koen De Bosschere. Towards Automatic Program Partitioning. In *Proceedings of the 6th ACM Conference on Computing Frontiers (CF)*, pages 89–98, New York, NY, 2009.

[SP01] Ram Subramanian and Santosh Pande. A Framework for Performance-based Program Partitioning. In book *Progress in Computer Research*, Nova Science Publishers, Inc., pages 151–169, Commack, NY, 2001.

[Vah02] Frank Vahid. Partitioning Sequential Programs for CAD using a Three-step Approach. *ACM Trans. Design Autom. Electr. Syst.*, 7(3):413–429, 2002.