# Accelerator-Based implementation of the Harris Algorithm*

Claude Tadonki[1], Lionel Lacassagne[2], Elwardani Dadi[3], and Mostafa El Daoudi[3]

[1] Mines ParisTech - Centre de Recherche en informatique, Mathématiques et systèmes 35, rue Saint Honoré, 77305 Fontainebleau Cedex, France
claude.tadonki@mines-paristech.fr
[2] Institute of Fundamendal Electronics, University of Orsay, Faculty of Sciences, Bat. 220, 91405 Orsay Cedex, France
[3] Université Mohamed Premier Oujda, Boulevard Mohamed VI, Oujda, Morocco

**Abstract.** Real-time implementations of *corner detection* is crucial as it is a key ingredient for other image processing kernels like *pattern recognition* and *motion detection*. Indeed, *motion detection* requires the analysis of a continuous flow of images, thus a real-time processing implies the use of highly optimized subroutines. We consider a tiled implementation of the Harris corner detection algorithm on the CELL processor. The algorithm is a chain of local operators applied to each pixel and its periphery. Such a special memory access pattern clearly exacerbates on the hierarchy transition penalty. In order to reduce the consequent time overhead, tiling is a commonly considered way. When it comes to *image processing filters*, incoming tiles are overdimensioned to include their neighborhood, necessary to update border pixels. As the volume of "extra data" depends on the tile shape, we need to find a good tiling strategy. On the CELL, such investigation is not directly possible with native DMA routines. We overcome the problem by enhancing the DMA mechanism to operate with non conventional requests. Based on this extension, we proceed with experiments on the CELL with a wide range of tile sizes and shapes, thus trying to confirm our intuition on the optimal configuration.

**Keywords:** Accelerator, CELL BE, Harris, image processing, tiling, DMA.

## 1 Introduction

The common characteristic of image processing algorithms is the heavy use of basic operators. Indeed, the typical scheme is a repetitive application of

* Work jointly supported by ANR projects Ocelle and PetaQCD, also by the Excellence Grant of Moroccan Ministry of Higher Education. Grant No. G 08/004.

linear and local kernels at the pixel level. The fact that each output pixel is obtained from the corresponding input pixel and it periphery breaks any hope of regular memory access, thus making it hard to achieve real-time performance implementations.

The *Harris* algorithm [4] for corner detection is an interesting case study application because it allows various implementation and optimization strategies [6]. Among these possibilities, tiling [10] is potentially attractive as it can be considered inside any valid scheduling as a an additional (memory) optimization. However, tiling on the CELL cannot be directly implemented because of data alignment constraints when using native DMA routines. Because of this constraint, tiles corresponding to contiguous memory region (full row tiles for instance) are used most of the time. Thus, their is no choice for the tile shape.

Tile shape restriction is particularly frustrating with image processing operators because either it does not allow the use of a predicted optimal tile shape, or it acts as a runtime bottleneck. The later could occurs, for instance, with an image so large that the SPE local store cannot hold three of its entire rows (one active row plus its top and bottom neighborhoods). *Data alignment* is another critical requirement. On this paper, we focus on the problem and provide a seamless effective solution. We study the effect of tiling and report experimental results driven by theoretical predictions. Our approach is more general for an accelerated-based computation, we chose CELL BE to illustrate our strategy.

The rest of the paper is organized as follows. The next section presents an overview of the CELL Broadband Engine. We describe the *Harris-Stephens algorithm* and some implementation considerations in Section 3. In Section 4, we discuss about tiling and predict the optimal tile shape. This is followed in Section 5 by an outline of the DMA issue when consider general tile shapes and what we provide to overcome the problem. Section 6 presents and analyses our experimental result according to our predictions. Section 7 concludes the paper.

## 2 The CELL Broadband Engine

Designed to provide a real-time processing response and a high-bandwidth network, the CELL [1,5] is a multi-core chip composed of nine processing elements. One core, the *POWER Processing Element* (PPE), is a 64-bit Power Architecture acting as a kind of master. The remaining eight cores, the *Synergistic Processing Elements* (SPEs), RISC architecture with SIMD organization with 128-bit vector registers and 256 KB of local memory, referred to as local store (LS). Each SPE has a clock speed of 4 Ghz (3.2 Ghz in average), with a peak performance of 256 GFlops (single precision) and 26 GFlops (double precision). The chip can handle 128 concurrent transactions to memory per processor. Figure 1 provides a synthetic view of the CELL architecture [5].

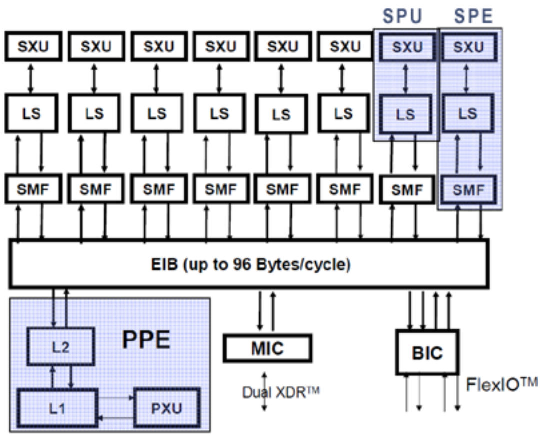


**Fig. 1.** Cell Chip Block Diagram

Programming the CELL is mainly a mixture of single instruction multiple data parallelism, instruction level parallelism and thread-level parallelism. The chip was primarily intended for digital image/video processing, but was immediately considered for general purpose scientific programming (see [9] for an exhaustive report on the potential of the CELL BE for several key scientific computing kernels). A specific consideration for QR factorization is presented in [2]. Nevertheless, exploiting the capabilities of the CELL in a standard programming context is really challenging. The programmer has to deal with hardware and software constraints like *data alignment, local store size, double precision penalty, different level of parallelism.* Efficient implementation on the CELL is commonly a conjunction of a good computation/DMA overlap and a heavy use of the SPU intrinsics.

## 3 The Harris-Stephen Algorithm

*Harris and Stephen [4] interest point detection algorithm* is an improved variant of the *Moravec corner detector* [7], used in computer vision for feature extraction like *motion detection, image matching, tracking, 3D reconstruction* and *object recognition.* Figure 2 illustrates the use of the algorithm.

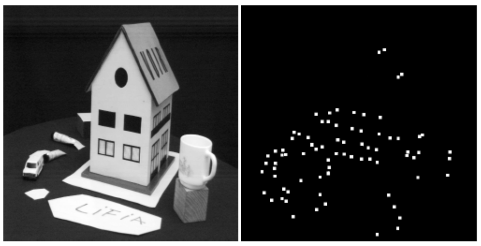


**Fig. 2.** Illustration of the Harris-Stephens procedure

The algorithm is mainly a succession of local operators implementing a discrete form of an autocorrelation $S$, given by

$$S(x,y) = \sum_{u,v} w(u,v)[I(x,y) - I(x-u, y-v)]^2, \tag{1}$$

where $(x,y)$ is the location of a pixel with color value $I(x,y)$, and $u, v \in 1, 2, 3$ model the move on each dimension. At a given point $(x,y)$ of the image, the value of $S(x,y)$ is compared to a suitable *threshold*, and the decision follows on the nature of the pixel at $(x,y)$. Roughly speaking, the process is achieved by applying four discrete operators, namely *Sobel* (S), *Multiplication* (M), *Gauss* (G), and *Coarsity* (C). Figure 3 displays an overview of the global workflow.

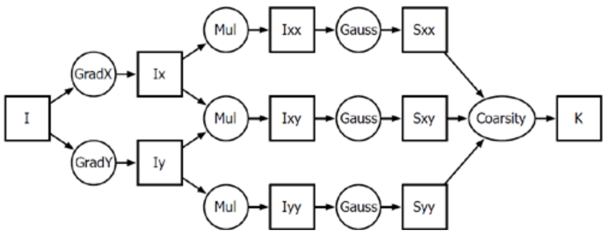


**Fig. 3.** Harris algorithm diagram

*Multiplication* and *Coarsity* are point to point operators, while *Sobel* and *Gauss*, which approximate the first and second derivatives, are $9 \to 1$ or $3 \times 3$ operators defined by

$$S_x = \frac{1}{8}\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad S_y = \tfrac{1}{8}\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \tag{2}$$

$$G = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} \tag{3}$$

Applying a $3 \times 3$ operator to a given pixel $(x,y)$ consists in a point-to-point multiplication of the corresponding $3 \times 3$ matrix by the following pixels matrix

$$\begin{pmatrix} I(x-1,y-1) & I(x-1,y) & I(x-1,y+1) \\ I(x,y-1) & I(x,y) & I(x,y+1) \\ I(x+1,y-1) & I(x+1,y) & I(x+1,y+1) \end{pmatrix} \tag{4}$$

Here comes the notion of *border*. In order to compute an output pixel $O(x,y)$, we need the pixel $I(x,y)$ and its immediate periphery. We say the operator is of *depth* 1. Operator depth is additive, means that if two operators $f$ and $g$ are of depth $p$ and $q$ respectively, then the depth of $f \circ g$ is $p + q$. Three problems are raised by the way operators are applied:

- accessing the points at the periphery yields an irregular memory access pattern, which is a serious performance issue
- computing two consecutive points involves some reused pixels (those on their common border). This yields a memory access redundancy, thus another performance issue
- applying each operator separately implies several read and write operations on the main memory (same location or not), yet another source of performance penalty

There are several ways to deal with the above problems. One way is to fuse or chain consecutive operators whenever possible. This overcome the repetitive read/write of the entire image, at the price of data and computation redundancy (more border pixels), thus should be done under a certain compromise. The first two issues are well tackled by tiling, which could be considered with fused operators. Although tiling is a more general technique, we really need a specific analysis in order to understand how the extra data that covers each incoming tile affect the global performance when dealing with operator-based algorithms.

## 4 Tiling Consideration

When applying an operator to a given tile, we need some *extra pixels* for the calculation of *border pixels*. This means that, applying the *Sobel* operator to a $a \times b$ tile yields a $(a-1) \times (b-1)$ tile. This aspect is usually referred in the reverse side, means that we require a $(a+k) \times (b+k)$ tile in order to produce a $a \times b$ tile, where $k$ is the *depth* of the operator. Redundant reads/writes and computations occur within the *border*, whose the volume depends on the tile shape. Indeed, since $(a+k) \times (b+k) = ab + k(a+b) + k^2 \approx ab + k(a+b)$, we see that the volume of the *border* is $k(a+b)$ for a $a \times b$ tile. Here comes the question about the optimal tile shape for a fixed tile volume (typically derived from memory constraints). We give the answer in proposition 1.

**Proposition 1.** *The optimal tile shape over the set of tiles with equal volume is a square tile.*

*Proof.* We need to minimize

$$M(a,b) = (a+b)\frac{W \times H}{ab}, \tag{5}$$

where $ab = \lambda$ (constant). Indeed, $\frac{W \times H}{ab}$ is the number of $a \times b$ tiles on the $W \times H$ region, and the border of a $a \times b$ tile is proportional to $(a+b)$. Reporting $b = \frac{\lambda}{a}$ in (5) yields

$$M(a,\lambda) = (a + \frac{\lambda}{a})\frac{W \times H}{\lambda}, \tag{6}$$

and we get

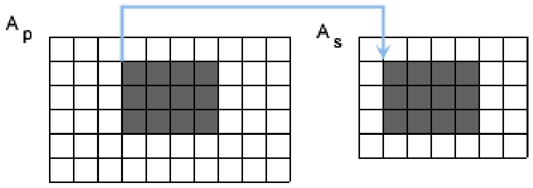
$$\frac{\partial M}{\partial a} = (1 - \frac{\lambda}{a^2})\frac{W \times H}{\lambda}. \tag{7}$$

Thus, $\frac{\partial M}{\partial a} = 0$ gives $a = \sqrt{\lambda}$ and then $b = \frac{\lambda}{\sqrt{\lambda}} = \sqrt{\lambda}$, i.e. $a = b$.

This result is important, provided the possibility to use any expected tile shape. We made this possible by encapsulating the necessary DMA into a single and generic routine. The result is general, but we need to check it for the case of the CELL because of the special access to the main memory. We use a scalar implementation of the operators and consider a full fused form of the Harris-Stephens algorithm.

## 5 DMA Issue with Standard Tiles

The problem we want to solve can be stated as follows. Given $M_p$, a $n_p \times m_p$ matrix on the main memory, and $M_s$, a $n_s \times m_s$ matrix on the local store. We need to copy the $a \times b$ submatrix of $A_p$ located at $(i_p, j_p)$ into $A_s$ at the location $(i_s, j_s)$. Figure 4 depicts the task.



Main memory: $n_p = 6$, $m_p = 10$, $i_p = 2$, $j_p = 4$
Local store: $n_s = 5$, $m_s = 7$, $i_s = 2$, $j_s = 2$
$a = 3$, $b = 4$

**Fig. 4.** Generic DMA pattern

Performing the transfer expressed in figure 4 raises number of problems:

- the region to be transfered is not contiguous on memory, thus list DMAs are used most of the time
- the address of one given row is not aligned, thus the global list DMA is not possible
- the (address, volume) pair of a row does not match the basic DMA rules (the above two ones), thus the entire list DMA cannot be carried out
- misalignment could come from both sides (main memory and/or local store)
- the target region on the local store might be out of the container limits

It is important to overcome the above problems at the minimum cost, since the consequent (pre/post)processing is an overhead for the programmer. To do so, we encapsulate all necessary (pre/post)processing into a single generic DMA subroutine. Roughly speaking, we perform either a direct DMA or a list DMA (one DMA per line of the tile), taking care of the above issues.

## 6 Experimental Results

We now proceed to some experimentations. The goal is to validate our implementation over various tile shapes, and see how close we are regarding our prediction on the optimal tile shape. Our program runs from the PPE and cooperate with one SPE. For each image, we chose a fixed tile volume and iterate on various shapes.

**Table 1.** Timings on a 512× 512 image

| $tile_h$ | $tile_w$ | total time($s$) |
|---|---|---|
| 8 | 512 | 0.0494 |
| 16 | 256 | 0.0598 |
| 32 | 128 | 0.0485 |
| 64 | 64 | 0.0345 |
| 128 | 32 | 0.0517 |
| 256 | 16 | 0.0699 |
| 512 | 8 | 0.0734 |

**Table 2.** Timings on a 2048× 512 image

| $tile_h$ | $tile_w$ | total time($s$) |
|---|---|---|
| 8 | 512 | 0.198 |
| 16 | 256 | 0.238 |
| 32 | 128 | 0.187 |
| 64 | 64 | 0.110 |
| 128 | 32 | 0.180 |
| 256 | 16 | 0.218 |
| 512 | 8 | 0.352 |

**Table 3.** Timings on a 1200× 1200 image

| $tile_h$ | $tile_w$ | total time($s$) |
|---|---|---|
| 5 | 1200 | 0.494 |
| 10 | 600 | 0.360 |
| 20 | 300 | 0.264 |
| 40 | 150 | 0.235 |
| 80 | 75 | 0.183 |
| 160 | 37 | 0.247 |
| 320 | 18 | 0.275 |

**Table 4.** Timings on a 2048× 2048 image

| $tile_h$ | $tile_w$ | total time($s$) |
|---|---|---|
| 8 | 512 | 0.985 |
| 16 | 256 | 0.726 |
| 32 | 128 | 0.643 |
| 64 | 64 | 0.438 |
| 128 | 32 | 0.692 |
| 256 | 16 | 0.866 |
| 512 | 8 | 1.422 |

We see that the most squared tile always gives the best global performance. The difference is marginal with closest shapes, but we should keep in mind that the typical use of the algorithm is with a flow of images. Our implementation does not overlap DMA with computations because of memory postprocessing due to misalignment. For wider images (Figures 3 and 4), we see that the improvement is more than 50% compared to full row tiles. We emphasize on the extra cost for managing irregular DMAs, although our implementation seems to perform well. The main difference between full row tiles and the others is that, for the later, DMA list is always necessary. Thus, the compromise here is between irregular DMAs and redundancies. Our experimental results clearly show that it still advisable to consider tiles with balanced dimensions.

## 7 Conclusion

The Harris-Stephens algorithm is a classical procedure in computer vision. From a programming point of view, it offers a wide range of optimization possibilities, each of them being appropriate for specific architecture. Since the CELL processor suits for image/video processing, investigating on the implementation of the Harris-Stephens algorithm is quite relevant, having in mind the impact on a stream processing context. In our work, we consider a tiled implementation based on a fused version of the algorithm. Using on our implementation of "irregular" DMAs, we provide a blocked implementation of the algorithm and we validate the optimal tile shape prediction. For absolute performances, we need to optimize our implementation of the basic operator (mainly with SPU intrinsics) and study how to overlap DMAs and computations. Due to the current status of the CELL BE, we plan to test our method on GPUs, and then consider the aforementioned issues in a more global context.

## References

1. Cell SDK 3.0, `www.ibm.com/developerworks/power/cell`
2. Kurzak, J., Dongarra, J.: QR factorization for the Cell Broadband Engine. Scientific Programming 17(1-2), 31–42 (2009)
3. `http://www.gnu.org/software/octave/`
4. Harris, C., Stephens, M.: A combined corner and edge detector. In: 4th ALVEY Vision Conference (1988)
5. Peter Hofstee, H.: Power Efficient Processor Design and the Cell Processor, `http://www.hpcaconf.org/hpca11/slides/Cell_Public_Hofstee.pdf`
6. Saidani, T., Lacassagne, L., Falcou, J., Tadonki, C., Bouaziz, S.: Parallelization Schemes for Memory Optimization on the Cell Processor: A Case Study on the Harris Corner Detector. HIPEAC Journal (2009)
7. Moravec, H.: Obstacle avoidance and navigation in the real world by a seeing robot rover. In: Tech. report CMU-RI-TR-80-03, Robotics Institute, Carnegie Mellon University & doctoral dissertation, Stanford University (September 1980)
8. Sen, S., Chatterjee, S.: Towards a theory of cache-efficient algorithms. In: SODA (2000)
9. Williams, S., Shalf, J., Oliker, L., Kamil, S., Husbands, P., Yelick, K.: Scientific Computing Kernels on the Cell Processor. International Journal of Parallel Programming (2007)
10. Xue, J.: Loop tiling for parallelism. Kluwer (2000)