Future Generation Computer Systems 88 (2018) 442–452

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Harris corner detection on a NUMA manycore☆

Olfa Haggui [a,1], Claude Tadonki [b,*], Lionel Lacassagne [c], Fatma Sayadi [d], Bouraoui Ouni [d]

[a] Sousse National School of Engineering, Networked Objects Control Communication Systems Lab (NOCCS), Tunisia
[b] Mines ParisTech - PSL Research University, Centre de Recherche en Informatique (CRI), France
[c] Sorbonne University, CNRS, Laboratoire d'Informatique de Paris 6 (LIP6), France
[d] Electronics and Microelectronics Laboratory, Faculty of Sciences, University of Monastir (FSM), Tunisia

HIGHLIGHTS

- Key points about real-time Harris on manycores.
- Consistent formulation of an efficient SIMD implementation.
- Complete discussion about cache memory considerations and related techniques.
- Shared memory parallelization and scalability study.
- Non-Uniform-Memory-Access (NUMA) adaptation.

ARTICLE INFO

Article history:
Received 27 September 2017
Received in revised form 15 January 2018
Accepted 23 January 2018
Available online 31 January 2018

Keywords:
Corner detection
Harris algorithm
Multicore
Multithread
SIMD
GFLOPS
NUMA
Scalability

ABSTRACT

Corner detection is a key kernel for many image processing procedures including *pattern recognition* and *motion detection*. The latter, for instance, mainly relies on the corner points for which spatial analyses are performed, typically on (probably live) videos or temporal flows of images. Thus, highly efficient *corner detection* is essential to meet the real-time requirement of associated applications. In this paper, we consider the corner detection algorithm proposed by Harris, whose the main work-flow is a composition of basic operators represented by their approximations using 3×3 matrices. The corresponding data access patterns follow a stencil model, which is known to require careful memory organization and management. Cache misses and other additional hindering factors with NUMA architectures need to be skillfully addressed in order to reach an efficient scalable implementation. In addition, with an increasingly wide vector registers, an efficient SIMD version should be designed and explicitly implemented. In this paper, we study a direct and explicit implementation of common and novel optimization strategies, and provide a NUMA-aware parallelization. Experimental results on a dual-socket INTEL Broadwell-E/EP show a noticeably good scalability performance.

1. Introduction

The common characteristic of image processing algorithms is the heavy use of convolution kernels. Indeed, the typical scheme is an iterative application of a stencil calculation at the pixel level. This yields non-local and unaligned memory accesses, thus making it hard to achieve a real-time performance implementation.

Harris corner (and edge) detection [1,2] is an important kernel in image processing, especially for motion detection and object recognition/detection/tracking [3,4]. Roughly speaking, the procedure is a serial combination of 3×3 filters (*derivatives* and *gaussians*), surrounded by basic arithmetic and selection operations. This leads to a *stencil computation* which exposes two correlated challenges concerning *memory accesses* and *redundant computation*. Since this kernel is likely to be called intensively on image processing applications, which includes the embedded context, fastest (at least real-time) implementations are crucial, hopefully on various hardware targets.

A thorough implementation study is provided by Lacassagne et al. in [5], where some of the basic ideas considered in this paper are mentioned, especially *arithmetic operations optimization* using the separability property of the filters, *computation reduction*,

☆ Work supported by the Carnot M.I.N.E.S project PACA, France.
* Corresponding author.
E-mail addresses: olfa.haggui@gmail.com (O. Haggui), claude.tadonki@mines-paristech.fr (C. Tadonki), lionel.lacassagne@lip6.fr (L. Lacassagne), sayadi-fatma@yahoo.fr (F. Sayadi), bouraoui.ouni-bouraoui@yahoo.fr (B. Ouni).
[1] Work done during the visit of Olfa Haggui at the Centre de Recherche en Informatique (CRI) - France.

https://doi.org/10.1016/j.future.2018.01.048
0167-739X/

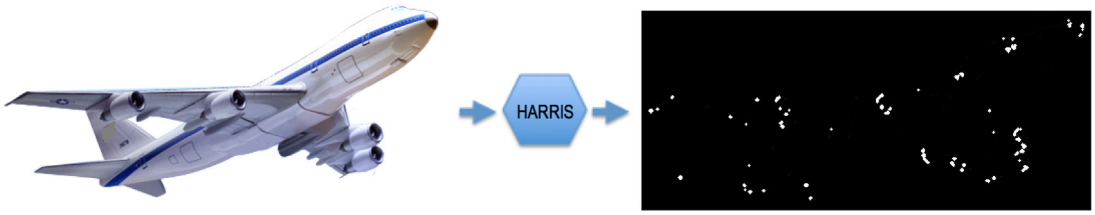

Fig. 1. Illustration of the Harris–Stephens corner detection.

memory accesses optimization through continuous data reuses, *loop collapsing* [6] of the operators in order to reduce the lifetime of intermediate data , and *array contraction* [7–10] which takes advantage of the shorter lifetime of intermediate variables to consider a compact storage through memory location reassignments. The SIMD part is left to the compiler (the native one for each considered architecture), and shared memory parallelism is implemented with OpenMP (through classical directives). Other studies of Harris corner detection and its applications can be found in [11–15].

In this paper, we follows the aforementioned basic ideas, but directly carried on in the SIMD context. Indeed, we consider $(i-1, j-1) \leftarrow (i, j)$ reindexation (as Kung–Lo–Lewis in [16] for the algebraic path problem) for each convolution and thereby obtain a perfectly aligned vector loads and stores at all levels of the loops. *Data reuse*, *factorization of the computations* and *optimal vector loads* are achieved through skillful registers shuffling and pipelining.

Regarding parallel implementation, we focus on the shared memory paradigm and basically apply a row-strip distribution of both input and output memory spaces among the working threads, each of them having its private space for intermediate values. We use Pthreads in order to have a full control of tasks allocation, data partitioning and thread/core association (we prevent thread migration). The main reason of this is that we are running on a NUMA architecture where any unaware memory scenario can incur a severe penalty [17–19]. We explicitly consider on-node memory allocations together with threads binding routines to implement an appropriate static partitioning, which minimizes remote accesses and internode bus (QPI) contention.

Clearly, besides the optimization techniques mentioned in [5], which include common ones and original ones, the novelty of this paper is twofold: a direct SIMD scheduling of the computation considering similar optimization techniques enhanced with efficient data management strategies and a NUMA-aware parallel implementation dedicated to unconventional multicores.

The rest of the paper is organized as follows. The next section (Section 2) provides a fundamental description of the Harris corner detection procedure, followed by an overview of the related work in Section 3. Section 4 introduces NUMA architectures and outlines the main related concerns. Our SIMD and SMP optimization strategies are described in Section 5. Performance results are presented and commented in Section 7. Section 8 concludes the paper.

2. The Harris algorithm for corner detection

Harris and Stephen [20] interest point detection algorithm is an improved variant of the *Moravec corner detector* [21], used in computer vision for *feature extraction*, *motion detection*, *image matching*, *tracking*, *3D reconstruction* and *object recognition*. An overview of the foundation of the algorithm, as described in [13] and similarly restated here, can be formulated as follows.

Let $I(x, y)$ denote the intensity of a pixel location (x, y) of the image, and λ a given threshold.

1. For each pixel (x, y) in the input image, compute the *Harris matrix* $G = \begin{pmatrix} g_{xx} & g_{xy} \\ g_{xy} & g_{yy} \end{pmatrix}$, with
$$g_{xx} = (\frac{\partial I}{\partial x})^2 \otimes w \quad g_{xy} = (\frac{\partial I}{\partial x}\frac{\partial I}{\partial y}) \otimes w \quad g_{yy} = (\frac{\partial I}{\partial y})^2 \otimes w, \tag{1}$$
where $\otimes$ denotes convolution operator and w is the Gaussian filter.
2. For each pixel (x, y), compute the Harris criterion given by
$$c(x, y) = det(G)k(trace(G))^2, \tag{2}$$
where $det(G) = g_{xx}.g_{yy}g_{xy}^2$, k an empirical constant, and $trace(G) = g_{xx} + g_{yy}$.
3. Set all $c(x, y)$ which are below λ to zero.
4. Extract points (x, y) having the maximum $c(x, y)$ within a window neighborhood. These points represent the corners.

Fig. 1 illustrates the result of the algorithm for the corner detection case.

The algorithm is mainly a successive application of convolution kernels that globally implement a discrete form of an autocorrelation S, given by

$$S(x, y) = \sum_{u,v} w(u, v)[I(x, y) - I(x-u, y-v)]^2, \tag{3}$$

where (x, y) is the location and $I(x, y)$ its intensity, and $u, v \in 1, 2, 3$ the components modeling 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* in order to determine the nature of the corresponding pixel. Roughly speaking, the process is achieved by applying four discrete operators, namely *Sobel* (S), *Multiplication* (M), *Gauss* (G), and *Coarsity* (C). Fig. 2 displays an overview of the global workflow.

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

$$S_x = \frac{1}{8}\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad S_y = \frac{1}{8}\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \tag{4}$$

$$G = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} \tag{5}$$

Applying a 3×3 operator H to a given pixel (x, y) consists in summing up its point-to-point multiplication with the following pixels matrix

$$\begin{pmatrix} I(x-1, y+1) & I(x, y+1) & I(x+1, y+1) \\ I(x-1, y) & I(x, y) & I(x+1, y) \\ I(x-1, y-1) & I(x, y-1) & I(x+1, y-1) \end{pmatrix} \tag{6}$$

Here comes the notion of *border*. In order to compute the output $H(x, y)$ for the pixel (x, y), we need its intensity $I(x, y)$ and those