Parallelization Strategies of the Canny Edge Detector for Multi-core CPUs and Many-core GPUs

Taieb Lamine Ben Cheikh, Giovanni Beltrame,
Gabriela Nicolescu and Farida Cheriet
Department of Computer Science
Ecole Polytechnique de Montréal
Montréal, Canada
Email: taieb.lamine-ibnecheikh@polymtl.ca

Sofiène Tahar
Department of Electrical and Computer Engineering
Concordia University
Montréal, Canada
Email: tahar@ece.concordia.ca

***Abstract*—In this paper we study two parallelization strategies (loop-level parallelism and domain decomposition), and we investigate their impact in terms of performance and scalability on two different parallel architectures. As a test application, we use the Canny Edge Detector due to its wide range of parallelization opportunities, and its frequent use in computer vision applications. Different parallel implementations of the Canny Edge Detector are run on two distinct hardware platforms, namely a multi-core CPU, and a many-core GPU. Our experiments uncover design rules that, depending on a set of applications and platform factors (parallel features, data size, and architecture), indicate which parallelization scheme is more suitable.**

I. Introduction

Parallel architectures are considered an efficient solution for the implementation of high-computation applications, such as computer vision software. This choice is motivated by (1) the use of large amounts of data, (2) the highly parallel nature of these applications, and (3) by the high computation capability of current multiprocessor architectures.

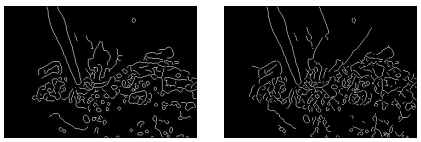
Among these architectures, we focus in particular on multi-core CPUs, and on many-core graphics processors (GPU). Multi-core CPUs offer great flexibility, and can execute code employing different parallelization strategies. However, current technology is limited to a small number of cores. Conversely, many-core GPUs provide a large number of cores, but their peak efficiency is limited to data parallel applications [1].

Since applications have increasing complexity and present a wide variety of parallel features, it becomes difficult to decide which parallelization strategy is suitable for a given architecture to reach peak performance. In this paper, we evaluate two different parallelization strategies, on two separate architectures: a multi-core CPU and a many-core GPU. We aim at determining guidelines for the efficient parallelization of common application classes.

We chose the Canny Edge Detector (CED) [2] as a benchmark for our experiments since it includes many common parallelism features, and it is widely used in computer vision applications. We compared four different parallel CED implementations on an AMD multi-core CPU and on an NVIDIA GPU.

The rest of the paper is organized as follows: Section II describes previous implementations of CED and some related work on parallelization; Section III introduces the Canny Edge Algorithm and the proposed parallelization strategies; In Section IV we show the evaluation results of each strategy; Finally, Section V draws some concluding remarks.

(a) Input Image (b) OpenCV Implementation [3] (c) Previous GPU Implementation [4]



(d) MATLAB Implementation [5] (e) Our Implementation

Fig. 1. Quality Comparison: output images for different CEDs, with the proposed implementation showing longer and smoother edges

II. Related Work

Domain decomposition and loop-level parallelism are well known techniques in parallel programming. Domain decomposition is used for solving partial differential equations on multiprocessor architectures [6] and for computer vision applications [7], while loop-level parallelism is a common strategy used by standards like OpenMP [8].

CED is used as a preprocessing phase in several computer vision applications. For example, CED has been used as a first step in the process of identifying and tracking instruments during laparoscopic surgery [9].

Several implementations of CED can be found in the literature, using different languages on different hardware platforms. These implementations generally lack accuracy or suffer from a long execution time. The CED included in the Intel OpenCV Libray [3] offers fast edge detection, but does not perform optimally in a noisy environment (see Fig. 1(b)). This lack of accuracy is caused by the use of relativly narrow filters with integer weights. Conversely, MATLAB [5] includes

978-1-4673-0859-5/12/$31.00 ©2012 IEEE

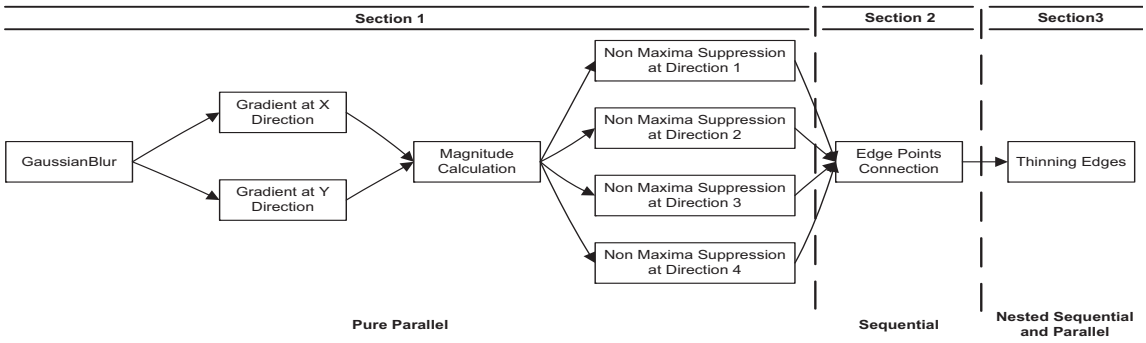


Fig. 2. Canny Edge Detector Diagram

an accurate Canny Edge Detector with larger filters and floating point weights, but requires longer execution times.

CED was implemented on a Tilera processor in [10], using loop-level parallelism and domain decomposition. The results of [10] show that domain decomposition offers better scalability compared to loop-level parallelism. However, their implementation is restricted to the Tilera64 architecture, and results are shown for only eight cores out of sixty-four available.

CED was also run on a NVIDIA GPU [4], using fine-grained loop-level parallelism. Although this last implementation offers the best performance in terms of execution time, it does not produce accurate and smooth edges as shown in Fig. 1(c).

The main challenge of all these implementations is to attain real-time execution, while keeping accurate edge detection. Our proposed implementation on GPU offers both high accuracyaround (e.g., for laparoscopic surgery) and low execution time as shown in Fig. 1(e) and Fig. 6, respectively.

In addition to this contribution, this paper describes two parallelization strategies that are generally not well studied for many-core GPUs. In the following, we provide a detailed performance evaluation of these strategies, for both multi-core CPUs and many-core GPUs.

III. Parallelization strategies

Similarly to the MATLAB version [5], our algorithm consists of 6 stages (see Fig. 2): (1) Gaussian Blur, (2) Gradient Calculation, (3) Magnitude Calculation, (4) Non Maxima suppression, (5) Connecting Edge Points, and (6) Thinning Edges. We can distinguish three sections in the algorithm, depending on the way in which parallelism can be exploited.

The first section is purely parallel, and can be equally distributed over different threads. The second section is strictly sequential, and the third consists of a sequential loop including purely parallel subsections. Based on this classification, we experiment with two different parallelization strategies.

A. Loop-level Parallelism vs. Domain Decomposition

Loop-level parallelism is considered a fine-grained technique: it consists of parallelizing loops where iterations can be executed independently. If one considers OpenMP, this operation is performed by adding specific directives, and the OpenMP runtime will be in charge of the allocation and distribution work to threads. Loop-level parallelism has a potentially higher reusability than domain decomposition: since CED includes a set of algorithms common to computer vision programs, such as Gaussian Blur and Gradient Calculation, their parallel implementation can be easily reused in other computer vision programs. However, loop-level parallelism may suffer from the overhead due to the thread's fork-join operations, and from poor data locality.

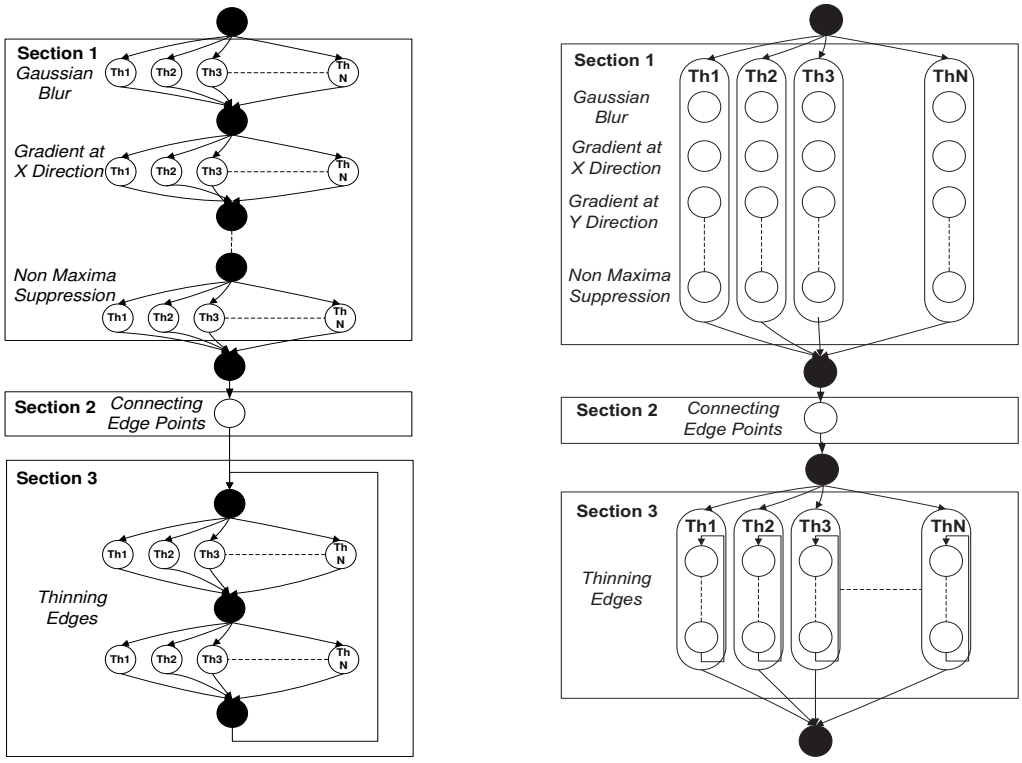
Conversely, domain decomposition is considered a coarse-grained technique, consisting in equally dividing an input data structure into sub data structures. In our study, the input image to the CED is split into sub images of equal sizes, and the original sequential program is executed on each sub image independently and in parallel. The main benefit of data decomposition is the ease of parallelization, since a large section of the code base is maintained, even though some additional code is needed to split and merge the data structures. It is also necessary to assess the presence of dependencies between neighbouring sub-images: in this case, the partitioning strategy has to be tuned to minimize the dependencies' impact on performance. Additional advantages of domain decomposition include avoiding the additional overhead caused by fork-join operations, and an increase in data locality.

The parallelization scheme for loop-level parallelism and domain decomposition is shown in Fig. 3(a) and Fig. 3(b), respectively.

IV. Experiments and Results

A. Implementation Details

The proposed CED is similar to that offered by MATLAB: we translated the MATLAB function to C++ and implemented it using OpenMP and CUDA [11], respectively, on a multi-core CPU and a many-core GPU. We applied our solutions to target images of different sizes (512x512 and 2048x2048) to evaluate the impact of data size on the scalability of the proposed implementations.



(a) Loop-level parallelism

(b) Domain Decomposition

Fig. 3. Parallelization Strategies

Our multi-core CPU platform is a dual AMD Opteron 6128, where each processor has 8 cores working at 2 GHz. For each processor, the cache memory is distributed as follows: 8 x 64 KB Data L1, 8 x 64 KB Instruction L1, 8 x 512 KB L2, and 10 MB shared L3. The system features 6 GB DDR3 memory at 1333 MHz for each processor.

Our GPU platform is the NVIDIA GeForce GTX 480, a Fermi-series graphics processor. This platform includes 480 Streaming Processors (SP) or cores distributed on 15 Streaming Multiprocessors (SM) as 32 SP per SM. Each core is working at 1.4 GHz, and the global memory assigned to the GPU is 1.5 GB.

B. Performance Evaluation

The CED was divided into three sections according to the parallelism opportunities provided by the algorithm (see Fig. 2). These sections were sequentially profiled using two images of different sizes, to identify the workload of each section, as illustrated in Table I. The profiling results show that: (1) the execution time of Section 2, the sequential section, is insignificant compared to the other parallel sections, thereby it has minor influence on the parallel scalability of CED and (2) most of the execution time takes place in Section 1 (increasingly with data size), which requires an efficient parallelization strategy for this section to reach optimal performance.

TABLE I
PROFILING OF CANNY EDGE DETECTOR'S SECTIONS ON CPU

Program Sections	512x512		2048x2048	
	Exec. Time	%	Exec. Time	%
Section 1	132	51.5	3889	72.4
Section 2	3	1.1	26	0.4
Section 3	118	46.1	1431	26.6
Total	256	100	5371	100

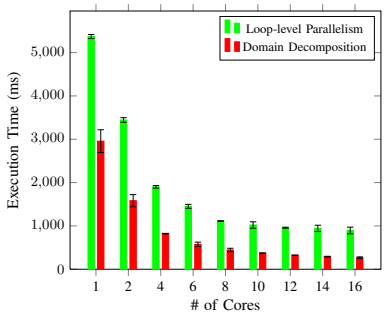


Fig. 4. Execution Profiling for 2048x2048 Image on a Multi-core CPU

1) Parallelization strategies on multi-core CPUs: We compared the speedup of domain decomposition and loop-level parallelism on a multi-core CPU: Fig. 4 shows that the domain decomposition strategy offers a considerable lower execution time than the loop-level parallelism strategy for large image sizes due to increased data locality. For small

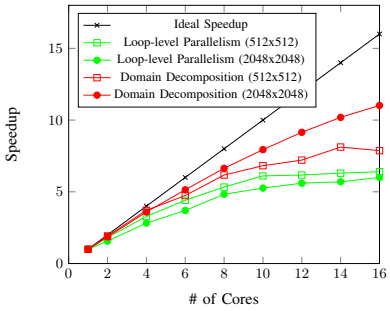


Fig. 5. Overall Speedup on a Multi-core CPU

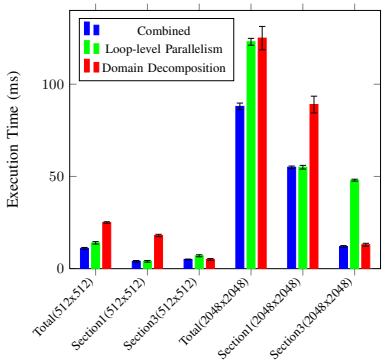


Fig. 6. Execution Profiling on a Many-core GPU

images (512x512) no considerable difference in the execution time could be found between the two strategies. As shown in Fig. 5 the domain decomposition provides better speedup and scalability than loop-level parallelism, and increasingly so for larger images.

2) Parallelization strategies on many-core GPUs: Concerning fine-grained parallelism, as expected, the application has a lower execution time for Section 1, as shown in Fig. 6. This is due to the fact that the GPU architecture is more suitable for fine-grained data parallelism, as opposed to multi-core CPUs [12]. Fig. 6 shows that, however, coarse-grained parallelism offers a lower execution time for Section 3. This is essentially due to the structure of the last section, which consists of an outer loop nested with a series of inner loops: each loop can be parallelized at pixel level, but each iteration of the outer while loop has to be executed sequentially. This means that the host CPU has to exchange data between the GPU and the CPU memory at each loop iteration, adding a significant overhead. Combining the two parallelization strategies under the parallelization strategy named *Combined* leads to the lowest execution time as shown in Fig. 6 . In addition, Fig. 7 shows that combining the two strategies guarantees the best speedup (33x when compared to sequential execution, and up to 3x with respect to 16 cores) for the 2048x2048 image sizes.

V. Conclusions

In this paper we showed that there is no "silver bullet" parallelization strategy for a given program. One parallelization

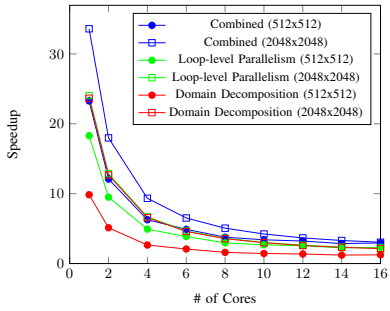


Fig. 7. Many-core GPU vs. Multi-core CPU

strategy may perform well in a given context (architecture, parallel features and data sizes), while it may be performing poorly in another. Based on the performance evaluation of the two studied parallelization strategies, we can conclude that loop-level parallelism works well for pure data parallel program running on GPUs, while domain decomposition is well suited for the processing of large amounts of data on multi-core architectures. As future work, we plan to evaluate the performance of different programming models for each of our strategies.

References

[1] S. Lee, M. M. T. Chakravarty, V. Grover, and G. Keller, "GPU kernels as data-parallel array computations in haskell," *Methods*, vol. 23, pp. 1–9, 2009.

[2] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679–698, 1986.

[3] G. Bradski, "The opencv library," *Dr Dobbs Journal of Software Tools*, vol. 25, no. 11, pp. 120–126, 2000. [Online]. Available: http://opencv.willowgarage.com

[4] Y. M. Luo and R. Duraiswami, "Canny edge detection on NVIDIA CUDA," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1–8, 2008.

[5] MATLAB, *version 7.10.0 (R2010a)*. Natick, Massachusetts: The MathWorks Inc., 2010.

[6] K. Sun, Q. Zhou, K. Mohanram, and D. C. Sorensen, "Parallel domain decomposition for simulation of large-scale power grids," *IEEE/ACM International Conference on Computer Aided Design*, no. 3, pp. 54–59, 2007.

[7] T. P. Chen, D. Budnikov, C. J. Hughes, and Y.-K. Chen, "Computer vision on multi-core processors: Articulated body tracking," *IEEE International Conference on Multimedia and Expo*, pp. 1862–1865, 2007.

[8] C. Terboven, D. An Mey, and S. Sarholz, "Openmp in multicore architectures," *International Workshop on OpenMP A Practical Programming Model for the MultiCore Era*, pp. 1–15, 2008.

[9] L. Windisch, F. Cheriet, and G. Grimard, "Bayesian differentiation of multi-scale line-structures for model-free instrument segmentation in thoracoscopic images," in *International Conference on Image Analysis and Recognition*, 2005, pp. 938–948.

[10] A. Z. Brethorst, N. Desai, D. P. Enright, and R. Scrofano, "Performance evaluation of canny edge detection on a tiled multicore architecture," in *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, vol. 7872, 2011, pp. 1–8.

[11] *NVIDIA CUDA Compute Unified Device Architecture - Programming Guide*, 2007.

[12] N. Bell, S. Dalton, and L. Olson, "Exposing fine-grained parallelism in algebraic multigrid methods," NVIDIA Technical Report, NVIDIA Corporation, June 2011.