



Stichting NIOC

Stichting NIOC en de NIOC kennisbank

Stichting NIOC (www.nioc.nl) stelt zich conform zijn statuten tot doel: het realiseren van congressen over informatica onderwijs en voorts al hetgeen met een en ander rechtstreeks of zijdelings verband houdt of daartoe bevorderlijk kan zijn, alles in de ruimste zin des woords.

De stichting NIOC neemt de archivering van de resultaten van de congressen voor zijn rekening. De website www.nioc.nl ontsluit onder "Eerdere congressen" de gearchiveerde websites van eerdere congressen. De vele afzonderlijke congresbijdragen zijn opgenomen in een kennisbank die via dezelfde website onder "NIOC kennisbank" ontsloten wordt.

Op dit moment bevat de NIOC kennisbank alle bijdragen, incl. die van het laatste congres (NIOC2023, gehouden op donderdag 30 maart 2023 jl. en georganiseerd door NHL Stenden Hogeschool). Bij elkaar bijna 1500 bijdragen!

We roepen je op, na het lezen van het document dat door jou is gedownload, de auteur(s) feedback te geven. Dit kan door je te registreren als gebruiker van de NIOC kennisbank. Na registratie krijg je bericht hoe in te loggen op de NIOC kennisbank.

Het eerstvolgende NIOC vindt plaats op donderdag 27 maart 2025 in Zwolle en wordt dan georganiseerd door Hogeschool Windesheim. Houd onze website (www.nioc.nl) in de gaten.

Wil je op de hoogte blijven van de ontwikkeling rond Stichting NIOC en de NIOC kennisbank, schrijf je dan in op de nieuwsbrief via

www.nioc.nl/nioc-kennisbank/aanmelden-nieuwsbrief

Reacties over de NIOC kennisbank en de inhoud daarvan kun je richten aan de beheerder:

R. Smedinga kennisbank@nioc.nl.

Vermeld bij reacties jouw naam en telefoonnummer voor nader contact.

Multi-core CPU/GPU

Accelerating Sequential Computer Vision Algorithms using Commodity Parallel Hardware

Door: Jaap van de Loosdrecht, Noordelijke Hogeschool Leeuwarden

Trefwoorden: parallelization of sequential code, Multi-core CPU/GPU, OpenMP, OpenCL

From 2004 onwards the clock frequency of CPU's has not increased significantly. The only way to get more performance is to go for parallel programming, which is not an easy way. This presentation was a result of a research master project at LIT in Ireland.

22 programming languages and environments for parallel computing on multi-core CPUs and GPUs are examined, compared and evaluated. OpenMP and OpenCL are chosen as standards and used to parallelize a number of standard and well-known algorithms in Computer Vision. The benefits and costs of parallel approaches to implementation of Computer Vision algorithms are evaluated.

'The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software', Sutter (2005)

Introduction

The last decade has seen an increasing demand from the industrial field for computerized visual inspection. However, applications rapidly become more complex and often with more demanding real time constraints. From 2004 onwards the clock frequency of CPUs has not increased significantly. Computer Vision applications have an increasing demand for more processing power and are limited by the performance capabilities of sequential processor architectures. The only way to get more performance using commodity hardware is to go for parallel programming.

There has been extensive research and development in parallel architectures and programming techniques for CPUs and Graphics Processor Units (GPUs). This paper summarizes an on-going research project (Van de Loosdrecht, 2013) that proposes to apply parallel programming techniques to meet the challenges posed in computer vision by the limits of sequential architectures.

This project has examined a number of different approaches and standards for exploiting parallel programming. One approach was chosen for multi-core CPUs and one for GPUs. These are integrated into the Computer Vision development environment VisionLab (VdLMV, 2013) and a detailed performance evaluation of the impact on a number of example algorithms have been undertaken to assess the benefits and costs of a parallel environment for computer vision.

Many other related research projects have been in a quest for one domain specific algorithm to compare the best sequential with best parallel implementation on a specific hardware platform. This project is distinctive because it investigates how to speed up a whole library by parallelizing the algorithms in an economical way and execute them on multiple platforms.

1. Survey of standards for parallel computing and programming

The motivation for this work is to research ways in which the large base of legacy sequential code of VisionLab could be accelerated using commodity parallel hardware such as multi-core processors and graphics cards. The VisionLab library is written in ANSI C++ and consists of more than 100,000 lines of source code. An important property of VisionLab proved to be that, because it is written in ANSI C++, it can be easily ported to different platforms like non PC based systems such as embedded real-time intelligent cameras and mobile systems. The compiler suites used are Microsoft Visual Studio and GNU.

A survey was performed on standards for parallel computing and programming. The most important conclusions are summarized in table 1 and table 2. For evaluating acceptance by the market the Evans Data Corporation survey in 2011 of the most popular multi-threaded APIs in North America discussed by Bergman (2011) was used. In this survey multi-threaded APIs for both CPUs and GPUs are ranked in one list.

OpenMP was chosen as the standard for multi-core CPU programming and OpenCL as standard for GPU programming. Portability, vendor independence and efficiently parallelizing the code were key decision factors. The OpenMP specification can be found at the home site of the OpenMP organization (OpenMP, 2011). The OpenCL specification can be found at the home site of the Khronos Group (Khronos Group, 2011).

2. Benchmarking

This work is about how to parallelize a large generic Computer Vision library in an efficient and effective way. The following classes of basic low level image operators were identified:

- Point operators;
- Local neighbor operators;
- Global operators;
- Connectivity based operators.

Low level image operators are used very frequently in many vision applications. The idea behind classifying these operators is that if a skeleton for parallelizing one representative in a class is implemented, this skeleton can be reused for the other representative in this class. For each class a representative operator was chosen and parallelization approaches were reviewed. The following basic low level image operators were implemented using OpenMP and OpenCL:

- Threshold;
- Convolution;
- Histogram;
- Connected component labeling.

A benchmark environment for assessing the benefits of parallelizing algorithms was designed and implemented. This benchmark environment was integrated in the script language of VisionLab. By using this benchmark environment it was possible to setup, run and analyze the benchmarks in a comfortable and replicable way. The reference for benchmarking is the sequential algorithm of VisionLab.

	Requirement								
Standard	Industry standard	Maturity	Acceptance by market	Future developments	Vendor independence	Portability	Scalable to ccNUMA (optional)	Vector capabilities (optional)	Effort for conversion
Array Building Blocks	No	Beta	New, not ranked	Good	Poor	Poor	No	Yes	Huge
C++11 Threads	Yes	Partly new	New, not ranked	Good	Good	Good	No	No	Huge
Cilk Plus	No	Good	Rank 6	Good	Reasonable No MSVC	Reasonable	No	Yes	Low
MCAPI	No	Poor	Not ranked	Poor	Poor	Poor	Yes	No	Huge
MPI	Yes	Excellent	Rank 7	Good	Good	Good	Yes	No	Huge
OpenMP	Yes	Excellent	Rank 1	Good	Good	Good	Yes, only GNU	No	Low
Parallel Patterns Library	No	Reasonable	New, not ranked	Good	Poor Only MSVC	Poor	No	No	Huge
Posix Threads	Yes	Excellent	Not ranked	Poor	Good	Good	No	No	Huge
Thread Building Blocks	No	Good	Rank 3	Good	Reasonable	Reasonable	No	No	Huge

Table 1. Comparison table for standards for Multi-core CPU programming (MSVC = Microsoft Visual C++, GNU = GNU C++ compiler)

	Requirement								
Standard	Industry standard	Maturity	Acceptance by market	Future developments	Expected familiarization time	Hardware vendor independence	Software vendor independence	Portability	Heterogeneous
C++ AMP	No	New	Not ranked	Unknown	Medium	Bad	Bad	Poor	No
CUDA	No	Good	Rank 5	Good	High	Reasonable	Reasonable	Reasonable	No
Direct Compute	No	Poor	Not ranked	Unknown	High	Bad	Bad	Bad	No
HMPP	No	Poor	Not ranked	Plan for open standard	Medium	Reasonable	Bad	Good	Yes
OpenCL	Yes	Reasonable	Rank 2	Good	High	Excellent	Good	Good	Yes
PGI Accelerator	No	Reasonable	Not ranked	Unknown	Medium	Bad	Bad	Bad	No

Table 2. Comparison table for standards for GPU programming

3. Evaluation of chosen standards.

3.1 Evaluation choice for OpenMP

Learning OpenMP was easy because there are only a limited number of concepts which have a high level of abstraction with only a few parameters. The used development environments, Visual Studio and the GNU tool chain, have a mature and stable implementation of OpenMP. OpenMP supports multi-core CPU programming but offers no support for exploiting the vector capabilities.

The effort for parallelizing embarrassingly parallel algorithms, like Threshold and Convolution, is just adding one line with the OpenMP pragma. More complicated algorithms like Histogram and Connectivity component labeling need more effort to parallelize. The effort for adding to the Automatic Operator Parallelization calibration procedure remains the same. Speedups between 2.5 and 5 were reported for large images in the benchmarks on a quad-core Intel I7 running Windows 7. Big penalties for speedup were reported in almost all benchmarks for small images. So run-time prediction whether parallelization is expected to be beneficial is a necessity.

The four basic low level image operators were used as templates to parallelize 170 operators of VisionLab, including many high level operators. It only took about two man months of work to parallelize 170 operators and to implement the run-time prediction mechanism.

VisionLab scripts written by users will, without modification, immediately benefit in speedup when using the new parallelized version of VisionLab. Users of VisionLab who write their code in C++ or C# will benefit, without changing their code, from the parallelization after linking to the new library. In two cases of real projects speedups were reported between 1.7 and 5 on a quad-core Intel I7 running Windows 7.

The portability of the OpenMP approach was tested on a quad-core ARM running Linux. Porting was just recompiling. It passed the VisionLab regression test suite without any problems and the Convolution benchmark reported speedups up to 3.97.

It is concluded that OpenMP is very well suited for parallelizing many algorithms of a library in an economical way and executing them with an adequate speedup on multi-core CPU platforms.

3.2 Evaluation choice for OpenCL

Although the author has a background in parallel programming, learning OpenCL was found difficult and time-consuming because:

- There are many concepts, often with a low level of abstraction and many parameters. Good understanding of GPU architectures is essential. The author had to master a new mindset.
- The logic of an algorithm is divided over the kernel language on the GPU and host language on the CPU with often subtle dependencies.
- The host-side code is labor intensive and sensitive for errors because most OpenCL API functions have many parameters.
- The kernel language itself is not difficult but there are many details to master.
- The correct tuning of many parameters is laborious but paramount for decent performance.

Instead of writing the host API code in C(++), VisionLab scripts were used. The script language of VisionLab was extended with OpenCL host API commands. Using these commands, it greatly reduced the time to develop and test the host-side code.

OpenCL supports both multi-core CPU and GPU programming. OpenCL also has support for exploiting the vector capabilities and heterogeneous computing.

The effort to parallelize embarrassingly parallel algorithms was considerable, both kernel code and host side code had to be developed. OpenCL versions for CPU and GPU were implemented for the four basic low level image operators. In many cases simple implementations demonstrated considerable speedups. In all cases a considerable amount of effort was necessary to obtain better speedups by making more complex algorithms and tuning parameters. For the Connected Component Labeling algorithm a complete new approach was necessary. For contemporary GPUs the overhead of data transfer between PC and graphics card is substantial compared to the kernel executing time of embarrassingly parallel algorithms like Threshold. When the new heterogeneous architectures hit the market, such as predicted by the HSA Foundation, this data transfer overhead will be reduced significantly.

Benchmarking was performed on a quad-core Intel I7 with a NVIDIA GeForce GTX 560 Ti graphics card running Windows 7. Speedups up to 60 were reported on benchmarks for large images. Big penalties for speedup were reported in some of the benchmarks for small images or if wrong tuning parameters were chosen. Completely different approaches were necessary for CPU and GPU implementations. Tests with the OpenCL Histogram implementations on NVIDIA and AMD GPUs suggest that GPU implementations for different GPUs need different approaches and/or parameterization for optimal speedup. It is expected that OpenCL kernels are portable but the performance will not be portable. In other words, when an OpenCL kernel is parameterized well with the host code it will run on many OpenCL devices, but the maximal performance on a device will be obtained only with a device specific version of the kernel and with tuned parameters.

It is concluded that OpenCL is not very well suited for parallelizing all algorithms of a whole library in an economical way and executing them effectively on multiple platforms. But OpenCL has the potential to unleash the enormous processor power of GPUs, the vector capabilities of CPUs and heterogeneous computing.

It is recommended that OpenCL is to be used for accelerating dedicated algorithms on specific platforms when the following conditions are met:

- The algorithms are computationally expensive;
- The overhead of data transfer is relatively small compared to the execution time of the kernels involved;
- It is accepted that a considerable amount of effort is needed for writing and optimizing the code;
- It is accepted that the OpenCL code is optimized for one device or sub optimal speedup is acceptable if the code should run on different similar devices.

3.3 New emerging standard

In 2013 a new standard, OpenMP 4.0 with 'directives for attached accelerators', is expected that will allow portable OpenMP pragma style programming on multi-core CPUs and GPUs. With the new OpenMP standard it will be possible to utilize vector capabilities of CPUs and GPUs.

Compared with OpenCL this new standard will allow multi-core CPU and GPU programming at a higher abstraction level than OpenCL. With the new OpenMP standard it is expected by the author that it will be much easier to program portable code, but the code will not be as efficient as programmed with OpenCL.

4. Product innovation

This work resulted directly in innovation of the commercially available product VisionLab.

- 170 operators were parallelized using OpenMP. Users of VisionLab can now benefit from parallelization without having to rewrite their scripts, C++ or C# code.
- OpenCL toolbox was added to the development environment. Users of VisionLab can now comfortably write OpenCL host side code using the script language and edit their kernels. The OpenCL host interface was implemented and tested for NVIDIA, AMD and Intel OpenCL platforms.

5. Conclusions

Computer Vision applications rapidly become more complex and often with more demanding real time constraints, so there is an increasing demand for more processing power. This demand is also accelerated by the increasing pixel resolution of cameras.

Using OpenMP it was demonstrated that many algorithms of a library could be parallelized in an economical way and adequate speedups were achieved on two multi-core CPU platforms. A run-time prediction mechanism that will test whether parallelization will be beneficial was successfully implemented for this OpenMP approach. With a considerable amount of extra effort, OpenCL was used to achieve much higher speedups for specific algorithms on dedicated GPUs.

Literature

- [1] Bergman, R., 2011. *AMD Fusion Developers Summit Welcome*. [pdf] : AMD Available at: <http://developer.amd.com/afds/pages/keynote.aspx> [Accessed 30 July 2011].
- [2] Khronos Group, 2011. *Open Standards for Media Authoring and Acceleration*. [online] : Khronos Group. Available at: <http://www.khronos.org> [Accessed 23 May 2011].
- [3] OpenMP, 2011. *The OpenMP® API specification for parallel programming*. [online] : OpenMP.org. Available at: <http://openmp.org> [Accessed 23 May 2011].
- [4] Van de Loosdrecht, J., 2013. *Accelerating sequential Computer Vision algorithms using commodity parallel hardware*. Draft thesis 29 April 2013. Final version of thesis is expected to be published in autumn 2013 at www.vdlmv.nl/thesis.
- [5] VdLMV, 2013. *VisionLab*. Demo version VisionLab available at: www.vdlmv.nl.

Wilt u reageren op dit artikel of deze presentatie? Neem dan contact op met:

Jaap van de Loosdrecht; Coördinator NHL; Centre of Expertise in Computer Vision; NHL University of Applied Sciences

j.van.de.loosdrecht@nhl.nl