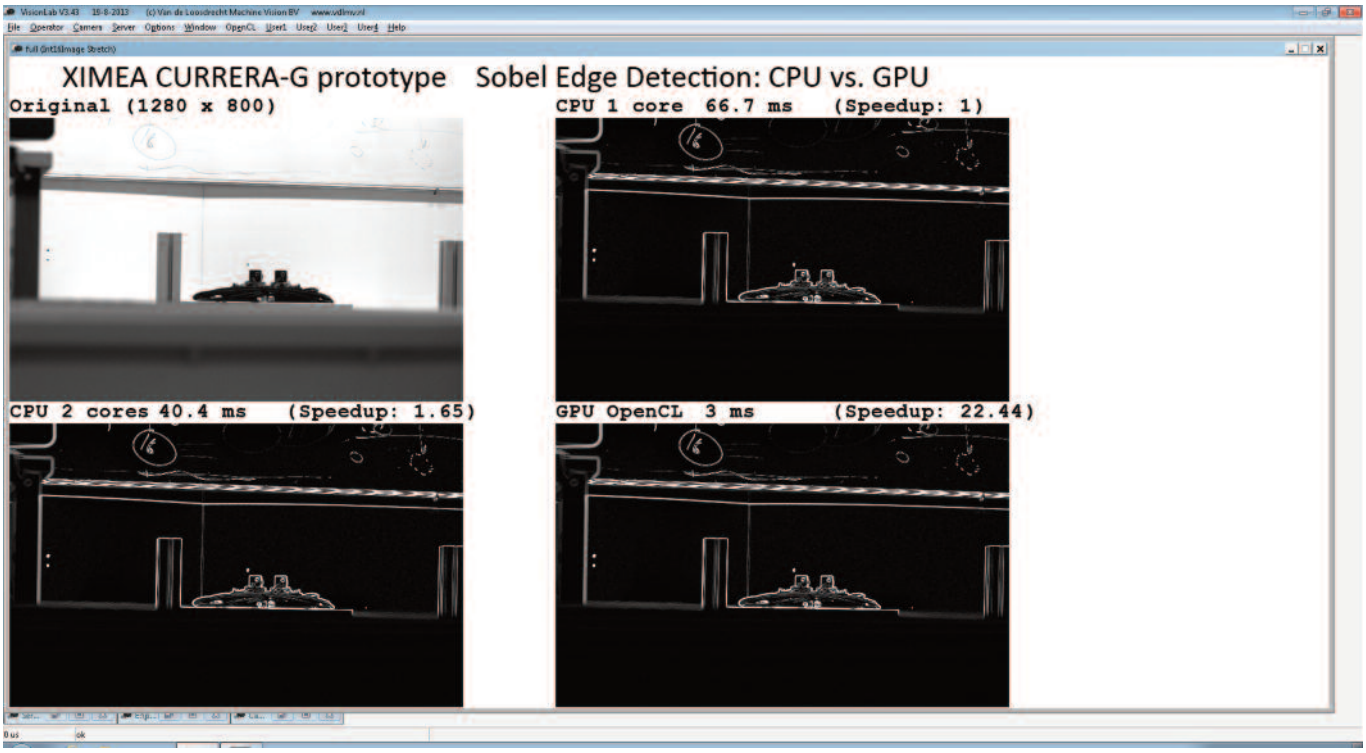


Bild: NHL University of Applied Sciences, The Netherlands

Bild 1 | Benchmark on intelligent camera with APU (=CPU+GPU): speedup on 2-core CPU=1.6x and speedup on 80-core GPU=22x . 1

# More Power

## Accelerating sequential Computer Vision algorithms using commodity parallel hardware

*The last decade has seen an increasing demand from the industrial field of computerized visual inspection. Applications rapidly become more complex and often with more demanding real time constraints. However, from 2004 onwards the clock frequency of CPUs has not increased significantly. Computer Vision applications have an increasing demand for more processing power but are limited by the performance capabilities of sequential processor architectures. The only way to get more performance using commodity hardware, like multi-core processors and graphics cards, is to go for parallel programming. This article focuses on the practical question: How can the processing time for vision algorithms be improved, by parallelization, in an economical way and execute them on multiple platforms?*

A survey [1] was performed on 22 standards for parallel computing and programming. OpenMP was chosen as the standard for multi-core CPU programming and OpenCL as the standard for GPU programming. Portability, vendor independence and efficiently parallelizing the code were key decision factors. The economical parallelization with respect to development effort and achieved speedups is researched by comparing it with an existing library. This VisionLab (VdLMV) library consists of over 100,000 lines of ANSI C++ sequential source code, making it platform independent. Parallelizing the complete library, in practice, would be too time consuming. The basic low level image operators can be classified in: Point operators, Local neighbour operators, Global operators and Connectivity based operators. Under the assumption that similar algorithms show similar speedups, one representative operator is chosen for each

| Algorithm | OpenMP | OpenCL |
|---|---|---|
| Threshold | 2.9 | 18.4 |
| Convolution | 4.9 | 60.9 |
| Histogram | 5.4 | 14.1 |
| Connected component labeling | 3.6 | 4.0 |

Table 1 | Speedups

class: Threshold, Convolution, Histogram and Connected component labeling. These operators are implemented in OpenMP 3.0 and OpenCL 1.1 and benchmarked. Programming effort is assed mainly by qualitative observation and speedup is measured by quantitative empirical research. The commodity hardware used for experimentation are an quad-core Intel I7 running Windows 7 with low-end NVIDIA and AMD graphics cards and a quad-core ARM running Linux. The Visual Studio and the GNU tool chains are used for programming the algorithms. The maximum speedups obtained for each of the four operators are shown in table. 1. The best speedups were obtained with large images. OpenMP extends C++ code with pragmas and functions in order to exploit parallelism. Learning OpenMP was considered easy because there are only a limited number of concepts which have a high level of abstraction. 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 overhead exceeds the speedup for small images. For this reason run-time prediction, whether parallelization is expected to be beneficial, is necessary. The four basic vision algorithms were used as templates to parallelize 170 operators. It only took about two man months of work to parallelize 170 operators and to implement the run-time prediction mechanism. Portability is achieved by just recompiling source code for a different platform. The OpenCL standard includes a language (based on C99) for writing kernels, functions that execute on the graphics card, plus an API that runs on the CPU for building, launching and controlling the kernels on the GPU. Learning and applying OpenCL was considered difficult and time-consuming. OpenCL introduces many new and low level abstract concepts. Although the OpenCL kernel language is easy to understand, high speedups can only be achieved if you understand all hardware related details. OpenCL can also be used for implementing algorithms on the CPU utilizing its vector capabilities. In the case of embarrassingly parallel algorithms simple implementations demonstrated considerable speedups. In all cases a considerable amount of effort was necessary to further improve the speedups by making more complex algorithms. Tests suggest that GPU implementations for different GPUs need different approaches for optimal speedup. It is expected that OpenCL operators are portable but their performance will not be portable to other graphics devices.

### *Conclusion*

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. With a considerable amount of extra effort OpenCL achieves much higher speedups for specific algorithms on dedicated GPUs. In the end the question is whether the speedup is worth the effort. The answer depends largely on the application. For the economic accelerating of a complete sequential computer vision library OpenCL is deemed unsuitable, but OpenCL has the potential to unleash the enormous processing power of graphics cards for specific application. For contemporary GPUs the overhead of data transfer between PC and graphics card is substantial compared to the kernel executing time. When the new announced heterogeneous CPU/GPU architectures hit the market, this data transfer overhead will be reduced significantly. The recent published OpenMP 4.0 standard will allow programming on both multi-core CPUs and GPUs. Compared with OpenCL, this new standard will allow multi-core CPU and GPU programming at a higher abstraction level than OpenCL. ■

[1]Thesis available at
www.vdlmv.nl/thesis

**www.nhl.nl/computervision**

Autoren | Jaap van de Loosdrecht & Klaas Dijkstra, Centre of Expertise in Computer Vision, NHL University of Applied Sciences, The Netherlands