Performance improvements for calculations of third party risk around airports

R. Aalmoes[1], R. Erkamp[2], Y. S. Cheung[1], R. van Nieuwpoort[2,3]
[1]*National Aerospace Laboratory NLR, The Netherlands*
[2]*VU University Amsterdam, The Netherlands*
[3]*Netherlands eScience Center, The Netherlands*

Abstract

During the past two decades, NLR has been performing research for third party risk around airports, and a calculation model and methodology for different types of airports are developed. The NLR third party risk model is used to evaluate the risk for people living and working around an airport. The design of new or changed air routes and runway infrastructure at airports require that third party risk studies are conducted to determine the impact for the airport surroundings. Due to the increase of traffic, the availability of improved individual flight track data, and the need for detailed calculations on a denser grid, the time needed for a risk calculation has increased significantly. It becomes challenging when more scenarios are involved and hence more risk calculations are needed in the decision making with respect to airport development and land use planning.

In this paper, we present the research of using parallel programming hardware to improve the performance of the calculation model, and in particular the use of graphics cards (GPUs) to carry out massive parallel operations. Because of different phases in the calculations, and the relation between adjacent grid cells, the translation of this model into a parallel implementation is not straightforward. Results show that a calculation for Schiphol airport, based on the traffic for a calendar year and a dense grid that took a week to complete in the original implementation, will finish well within one hour in the improved parallel implementation, with identical results. Other findings of the research show that the dedicated focus on improved performance has also helped finding and solving performance issues in the original model implementation.

Keywords: third party risk, performance optimization, airport, aerospace, GPGPU, OpenCL.

1 Introduction

The third party risk model is an analysis model for determining risk to third parties, thus the risk to the population around airports. The model is developed by National Aerospace Laboratory (NLR) of the Netherlands. The need for such a model arose in the last decades as the amount of air traffic increased and therewith the realisation of the population that they are exposed to risks because of air traffic. Since the airspace around airports is the most crowded, and the probability of an aircraft accident is the greatest in the take-off and landing phase (see Figure 1), the population in the vicinity of airports is exposed to the greatest risk. The increase in air traffic made the Ministry of Transport decide to have the NLR model the risk to third parties around airports. The model is used for multiple purposes, e.g. determining risk areas and evaluating development plans of airports (Weijts [1]). By introducing this model the Dutch government was aiming to implement their policy to increase the safety for individuals. In general, its objective is that nobody will have to live at a location where the individual risk is larger than 10^{-6}. This value includes the risks of other risk factors besides airports, such as the transport of hazardous substances (CBS [2]).

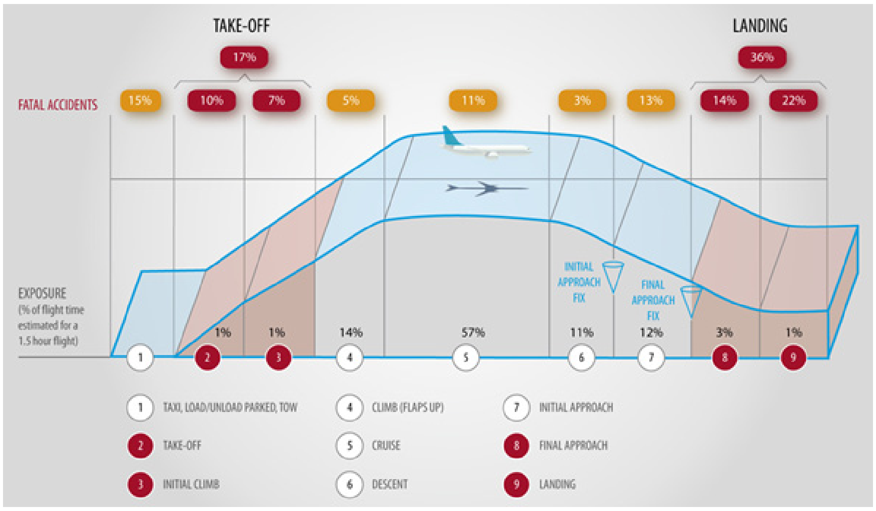

Figure 1: Percentage of fatal accidents by flight phase. Even though only 6% of a flight is spent in landing or take-off phase, most fatal accidents happen in these phases. (Source: Statistical Summary of Commercial Jet Airplane Accidents, 1959–2008, Boeing.)

In the last decades, the amount of air traffic has increased significantly. From a modelling perspective, this leads to more movements that should be taken into account. Also, an increase of study area may be required. Another development is that more detailed individual flight track data has become available: logging of individual radar tracks for each flight is more common and there is a demand to use this data to improve the accuracy of third party risk calculations. Finally, a greater accuracy has been demanded by users of the model results, meaning that calculations have to be done on a denser grid.

Each of these issues causes the required time to do a risk calculation to increase significantly. It becomes (even more) challenging when more scenarios are involved and hence more risk calculations are needed in the decision making with respect to airport development and land use planning. With the continuous performance improvements of computer hardware, the argument that computers are too slow for the requested improvements cannot keep ground any longer. We therefore make use of a multi-disciplinary approach, also known as *eScience*. There has been done work in the past to use graphics processors to speed up risk model implementations [3, 4], but not yet for the air transport domain. New model implementation methods should therefore be examined to make third party risk modelling ready for the next decade.

2 The third party risk calculation model

The third party risk model has a certain set of input parameters, on which internal processes operate to produce the requested output. The input parameters of the third party risk model can be split into two groups: model parameters and input data of a scenario. The model parameters are characteristics of the model and are derived from past events. The input data of a scenario is the data which characterizes the scenario for which the risk analysis should take place, like the number of aircraft movements, the aircraft types that are involved, and the routes they take. All this information will be used to calculate the risk for the scenario. The area for which calculations are requested is called the study area. The default study area for regional airports is 40 by 40 kilometres, with a cell width and cell height of 25 metres. For small airports, a smaller study area may be sufficient. For large airports, a larger study area may be needed. For example, for Schiphol Airport a study area of 56 by 56 kilometres is used.

The calculations that can be requested are Individual Risk (IR) and Societal Risk (SR). The Individual Risk is defined as the probability per year that an imaginary person permanently located on a fixed spot in the vicinity of an airport dies as a direct consequence of an aircraft accident (Weijts *et al.* [1]). The result of an IR calculation is a probability grid containing an IR value for each cell. The characteristics of this grid are the same as those of the study area as specified by the user. As an extra option, specific output can be requested. For example, one may request to compute additional grids containing the risk caused per accident type and per part of day (day and night). Requesting grids for specific subsets will result in extra grids in addition to the overall grid. The Societal Risk is defined as the probability per year that a group greater than N persons located in the study area die as a direct consequence of a single aircraft accident (Weijts *et al.* [1]). This paper focuses primarily on the individual risk. It is noteworthy that the challenges for optimising Societal Risk calculation are similar to those found for Individual Risk calculation.

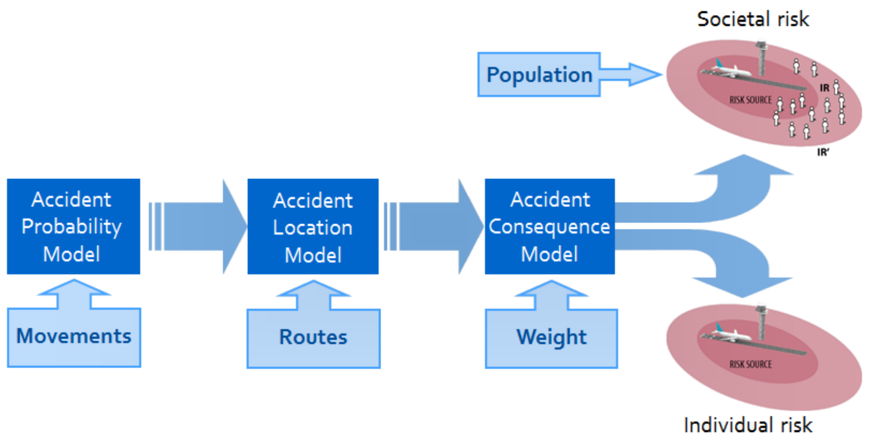

Figure 2: Flow chart of the sub models (source: NLR).

The IR and SR calculations require that the probability densities and the sizes of the crash areas are calculated in advance. This data is calculated by a number of sub models (see Figure 2): the Accident Probability Model (APM), the Accident Location Model (ALM), and the Accident Consequence Model (ACM).

The Accident Probability Model selects the accident rate (AR) of a certain accident type based on a couple of parameters: the aircraft's generation and maximum take-off weight, whether the flight is a cargo flight, passenger flight or a business jet, and whether the accident happens during take-off or landing. The accident rate is the probability that a certain accident occurs.

In the Accident Location Model, a probability density matrix is calculated for each accident rate selected in the Accident Probability Model by using one or more distribution functions. The probability density matrix is a grid with a probability density value for each cell. The probability density in a cell gives the probability that if an accident happens, the crash will occur in the cell. Each cell has its own probability density since its distance to the flight path or the runway threshold most likely differs. Selection of the right distribution function to use is based on the aircraft's maximum take-off weight, whether it is in take-off or landing phase, and the type of accident. Furthermore, the distribution function choice depends on whether the probability density matrix should be calculated relative to the route or relative to the runway (Weijts *et al.* [1]).

The Accident Consequence Model selects two parameters, i.e. crash area and lethality. The size of the crash area is based on the aircraft's maximum take-off weight. The crash areas are modelled as circles with its centre point located in the centre of a grid cell. Lethality is a constant and is dependent of the chosen model (Weijts *et al.* [1]).

2.1 Individual risk calculation

Computing the total risk caused by an aircraft accident in a certain cell depends on several factors. First, there is the probability that an accident actually occurs, also called the accident rate. Then, there is the probability that if the accident happens, the crash will occur in this cell. Note that this is the value in the probability density matrix. Two other factors are the size of the crash area and the lethality in the cell. The lethality is the probability of a person dying as a consequence of an aircraft accident when present in the crash area. The product of these four factors gives the total risk for a certain cell for a given accident type and movement (Weijts *et al.* [1]).

The risk calculated for one event applies for the complete crash area. Only when the crash area is entirely within the cell, the calculated risk is added to the cell's risk. If this is not the case, it is distributed over the cells within the crash area accordingly. Thus, for each cell (partially) covered by the crash area, the percentage of the crash area in that cell should be computed. Since the crash area is always modelled as a circle and the centre of that circle is always the centre of a cell, the same distribution can be applied to multiple cells. Therefore, for each crash area size, a template is computed containing the indices of the cells in the crash area, relative to the centre cell of the crash area, and the corresponding percentages (Weijts *et al.* [1]). See Figure 3 for an example crash area, its derived template, and the application of that template to the crash area in a grid. The total amount of templates that needs to be computed and stored is equal to the number of different crash area sizes. Computing a template only concerns the first quadrant of the crash area, because the values in the other quadrants are equal to the first only mirrored vertically, horizontally or diagonally. When, for every cell, all the risks are calculated and distributed around the involved cells, for each accident type, and for each movement, the Individual Risk calculation is finished.

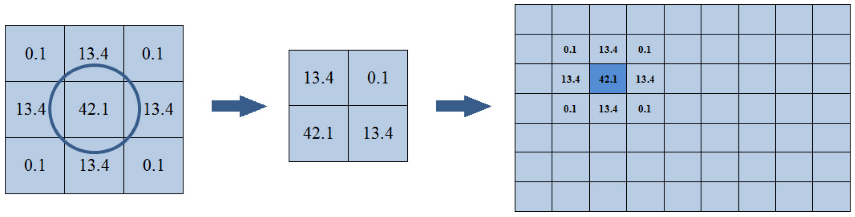

Figure 3: Derivation of a template from a crash area (blue circle), and application of that template to a grid.

3 The case for GPGPU and OpenCL

The Third Party risk model has been implemented in the past in a traditional, sequential program, suitable for a typical PC or workstation with a high performance Central Processing Unit (CPU), also known as the *processor*. But since the last decade, two hardware developments (Palacios and Triska [5]) have taken place that requires a re-evaluation for high-performance model implementations:

1. The computational power of graphics cards becomes not only useful for video games, but also for applications suitable for high parallelisation.

The use of graphics processors for this purpose is also called General-purpose computing on graphics processing units (GPGPU).

2. Due to thermal constraints, the clock speed of CPUs (a significant factor in increasing performance) reaches its limit. Consequently, manufacturers decide to provide CPUs with multiple calculation cores instead.

These developments mean for a sequential program (running on a single *core*), that it will only partially benefit from new generations of computing systems, and it will not benefit at all from the up-scaling by combining multiple cores in a single processor.

The processing power of a single CPU core is higher than that of a GPU core. But the number of cores in a CPU is still limited to about 8–12 at this moment, while the latest GPU generation (2012/2013) provides around 2500 cores [7]. If an application can make efficient use of these GPU cores, it can create a significant performance gain over an application only running on a CPU. If a specific application requires more cores than physically available, the GPGPU architecture takes care of scheduling the application in multiple batches for you [8, 9].

Another new development in the GPGPU field makes the use of this architecture for third party risk calculation finally suitable: the model implementation requires double-precision floating point mathematics, and only recent GPU architectures provide sufficient double precision processing power (Beyond3D [10]).

To develop GPGPU applications, two leading architectures exist: CUDA (Compute Unified Device Architecture) from NVIDIA, and OpenCL (Open Computing Language) from the Khronos group. Since CUDA is focused on NVIDIA GPUs, it can outperform OpenCL on these devices. A comprehensive performance comparison (Fang *et al.* [11]) shows that CUDA performs at most 30% better than OpenCL, but similar performance is achieved under a fair comparison (Fang *et al.* [11]). But because of its portability, especially the ability to run the application on a CPU (albeit slower in most cases than on a GPU), and its independence from a single manufacturer, we chose to use OpenCL to accelerate the program.

4 Design for a parallel third party risk implementation

Using GPUs may result in a significant performance improvement, but not all applications are fit for parallelisation. The way the model is implemented in software relates to the code instructions and the storage of the data that is being processed, and they will be discussed in the following sections.

4.1 Terminology

An implementation that makes use of the GPU will run partially on the CPU, also known as the *host*, and will delegate calculations that can be parallelised to the GPU, also known as the *device*. The GPU will distribute the calculation over the different cores to perform a task. A program running on the GPU is called a *kernel*. A task running on a single core is called a *work-item*.

4.2 Code considerations

A multi-core CPU can perform different tasks at the same time on different cores. In comparison, each GPU core will run the same kernel at the same time (Kreinin [6]). Different tasks should therefore be separated in time in order to run it on the GPU.

4.2.1 Instruction dependencies and branching

Some parts of a program cannot be executed in parallel, because the instructions rely on the result of other instructions. If the part of the program, in terms of execution time, that can be parallelised is small, overall performance gain, if any, will be small as well. This also depends on the size of the dataset on which the parallelised part operates. It would be ideal if this data set is large, in order to keep as many processing elements as possible busy.

Another important indication whether a parallel implementation is efficient, can be seen by branching in the execution path, which is incurred by conditional statements. The GPU can only execute one branch of the execution path at the same time. Imagine an if-statement in the kernel code for which just a couple of work-items satisfy the condition. A couple of processing elements have to execute the instructions for these work-items while the rest of the work-items are idle. Therefore, programs with execution path branching are less suitable for parallelisation than program without branching.

4.2.2 Selection of parallel code sections

The structure of the parallel program mainly adheres to the original program's structure. This means that at the highest level the program loops over all requested models. In other words, requested models are performed one by one. For each model, the program performs calculations movement by movement. Depending on the type of movement, multiple accident types may apply. For each accident type, the sequential program loops over all cells in the study area, performing either Individual Risk calculations or Societal Risk calculations for the concerning cell. This is the part of the program where most of the calculations happen and most time is spent. Therefore, these calculations are carried out concurrently in the parallel version (see Figure 4). The bold lines are executed concurrently in the parallel version.

We choose to do the calculations of multiple cells in parallel, rather than multiple movements. One reason is that if there are fewer movements than cells, it will lead to processor under-utilization. Another reason is that multiple work-items would need to update the same values, leading to race conditions.

```
FOR all models {
    FOR all movements {
        FOR all accident types applying to movement {
            FOR all cells in study area {
                Calculate risk value;
            }
        }
    }
}
```

Figure 4: Main structure of the sequential program. The bold lines are executed in parallel in the parallel version.

4.3 Data storage considerations

In comparison to the code that runs on each GPU core, the data that each core uses can be different. By requesting the GPU work-item identification, a different data set can be accessed for its calculations.

For the third party risk model, the risk values must be represented as double-precision floating point numbers. This requires extra design concern as important parallel functions like atomic operations are only supported in OpenCL for integer values, not double precision values. This situation may change in the future, as next generations of GPUs may support atomic operations for double precision.

The GPU also has different memory, and a different memory model. Transferring data from the CPU to the GPU and back requires some time, but can be done in parallel with other computations. Some work-items can also share a common memory area, called shared memory. By making use of this memory type, the third party risk implementation can be made more efficient.

4.4 Parallel Individual Risk implementation

In this article, not all implementation steps are explained in detail, but we show the considerations for parallelisation for one for the more important steps, the individual risk calculation. Important other steps, like the calculation of probability density matrices and the crash area size are discussed in Erkamp [15].

The Individual Risk calculation can be split into a calculation and a distribution phase. In the parallel version, each of those phases is implemented in a separate kernel. This technique of splitting up these phases is important to create a synchronisation point. This will be explained in the next sections.

4.4.1 Parallelising IR calculation phase

In the first phase, the risk caused by a movement, at a certain location, for a given accident type is calculated by the following formula: $IR_{(x,y)} = AR_{acc} \times PD_{(x,y)} \times CA_{(x,y)} \times$ lethality, where AR_{acc} is the accident rate of the given accident type, $PD_{(x,y)}$ is the probability density of the given accident type at location (x,y), and $CA_{(x,y)}$ is the crash area size of the accident type at location (x,y).

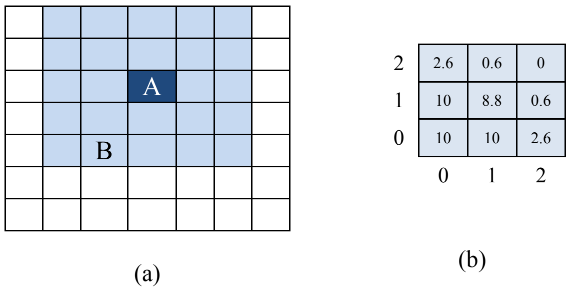

Figure 5: (a) Sample grid showing the neighbours which contribute to cell A’s risk. (b) (Upper-right) example distribution template of cell B.

The parallelisation of this phase is straightforward. Each work-item performs the multiplications itself. The accident rate, probability density, and lethality are provided by the host for each movement and accident type. Hence, at least one complete probability density matrix must be transferred for each movement, depending on the amount of applicable accident types.

4.4.2 Parallelising IR distribution phase

The second phase is the distribution of the risks, calculated in the first phase, over all cells being part of a cell’s crash area. The distribution is based on the percentage of the crash area that lies within the relevant cell, and is only dependent on the crash area, for reasons explained in Section 2.1. The same section showed an example of a distribution template.

In the sequential version, each cell’s calculated risk is immediately distributed over the relevant cells. This is not possible in the parallel version, because it would lead to race conditions, and atomic operations are not supported for double-precision floating points (see Section 4.3).

The solution to this problem is that each work-item should be allowed to update the risk value of its own cell only. Therefore, each cell should check its neighbouring cells for how much they contribute to its own risk value. This is done by multiplying a neighbour’s calculated risk value by the correct percentage from that neighbour’s distribution template (see Figure 5). The result of this multiplication is added the cell’s own risk value. Because we split up the calculation and distribution phase, we can be sure that the calculation has already taken place for all cells. This shows the necessity of the introduction of a synchronisation point in the implementation. This process is explained in more detail in Erkamp [15].

During the distribution phase, multiple work-items in a work-group (partially) require the same data located in global memory, including initial risk values of other cells and distribution templates. To reduce accesses to global memory, this data is read into shared memory.

Table 1: Timing comparison (in seconds) of third party risk model implementations on a representative Schiphol scenario.

	Sequential	Optimised Sequential	Parallel Tesla C2075	Parallel Tesla K20m
Probability density	8.99×10^3 s	8.99×10^3 s	839 s	474 s
Individual risk	515×10^3 s	62.9×10^3 s	384 s	163 s
Total	524×10^3 s	71.9×10^3 s	1.22×10^3	638 s

5 Results

5.1 Validation

The scenario that is calculated is a typical Schiphol scenario, with a year of representative, aggregated traffic movements. The output of all runs is validated by comparing it with the output of the sequential runs. These deviations are well within the limits set by previous validations [13, 14]. A partial validation can be found in Erkamp [15], and a complete validation for the third party risk model is still planned to take place.

5.2 Timing

The calculations for the comparison have been performed by a system with an Intel Core 2 E8400 CPU, which has two processing cores running at 3 GHz. The GPU in this machine is the NVIDIA Tesla C2075, which contains 6 GB of global memory and 448 processing cores running at 1.15 GHz. The timings have been split into the time required for the probabilities densities, and the time required for the individual risk calculation. This has been done for two reasons. First, the probability density calculation is only needed for the different movements. When the movements do not change, this step can be skipped. Second, the sequential model is only optimized for the individual risk calculation. Therefore, a more fair comparison can be made on the capabilities of the GPU if we only look at the individual risk comparison. For completeness, we also added the performance using a computer with the latest generation GPU; this is an Intel Xeon E5-2670, in which 8 processing cores run at 2.6 GHz. The GPU residing in this machine is NVIDIA's Tesla K20m, containing 5 GB of global memory and 2496 processing cores running at 705.5 MHz.

The results show that the probability density calculations are improved with a tenfold acceleration, and the individual risk calculation are increased by more than 1300, more than the number of cores in the Tesla C2075 (448). This indicates that the original implementation can be improved, and the resulting optimized sequential implementation is about seven times faster. However, the parallel individual risk calculation is still more than seventy times faster than this optimized version. For the complete calculation, the speed-up is over 58 times if we look at the optimised sequential implementation, and more than 400 for the original implementation. The latest generation Tesla K20m runs the calculation another two times faster than the Tesla C2075 system.

6 Conclusion

In this article, we described the NLR third party risk model and the need to improve the speed of the model implementation. The original model implementation was developed in the early 2000s, before the introduction of multi-core CPUs, or GPUs with general-purpose calculation possibilities. With the advent of these new computing technologies, this opened the prospect that a renewed model implementation would succeed in a significant speed-up. This research shows that the third party risk model is particularly suitable to be run with the help of a GPU, creating a 58 times speed-up. The new implementation is not straightforward, but with sufficient knowledge on how the model works, and what dependencies exist within the original model implementation, new GPU modelling techniques can be used to make full use of the potential of a GPU as generic calculation hardware.

This performance gain enables to compute risks on a dense grid (25 × 25 metre cells) in reasonable time, as preferred by the Dutch government. It also allows to compute risks with individual aircraft movements instead of aggregated movements, and to compare multiple scenarios with varying input parameters.

By performing a dedicated analysis of the original implementation, and creating an optimized GPU program, the research also showed the shortcomings in the original model implementation. Some of these improvements from the GPU program were translated back to the original sequential model application. This resulted in a tenfold performance gain. Although this is not yet near the performance of the GPU program, it shows the benefit of a focussed approach on improved performance. The combination of domain expertise (third party risk) and applied computer research on parallel computing has led to a successful new third party risk application that allows faster studies for airport safety. There is enough reason to believe that this multi-disciplinary approach, often called eScience, will work in other domains as well.

References

[1] Weijts, J., Vercammen, R.W.A., van de Vijver, Y.A.J.R. & Smeltink, J.W. *Voorschrift en procedure voor de berekening van Externe Veiligheid rondom luchthavens*. NLR-CR-2004-083, 2004.

[2] CBS, PBL, Wageningen UR (2007). *Externe veiligheidsrisico's: de kans op een ongeluk (inleiding)*. Indicator 0300, Ver. 05. September, 2007.

[3] Zhang, Y., Vouzis, P. & Sahinidis, N.V. GPU simulations for risk assessment in CO_2 geologic sequestration. *Computers & Chemical Engineering*. Volume 35, Issue 8, pages 1631-1644. August, 2011.

[4] Bahl, A.K., Baltzer, O., Rau-Chaplin, A., Varghese, B. & Whiteway, A. Multi-GPU Computing for Achieving Speedup in Real-time Aggregate Risk Analysis. *High Performance Computing on Graphics Processing Units (hgpu.org)*. February, 2013.

[5] Palacios, J. & Triska, J. *A Comparison of Modern GPU and CPU Architectures: And the Common Convergence of Both*. March, 2011.

[6] Kreinin, Y. *SIMD < SIMT < SMT: parallelism in NVIDIA GPUs.* http://www.yosefk.com/blog/simd-simt-smt-parallelism-in-nvidia-gpus.html. November, 2011.

[7] Tesla GPU Accelerators for Servers. NVIDIA. http://www.nvidia.com /object/tesla-servers.html

[8] Gaster, B.R., Howes, L., Kaeli, D. R., Mistry, P. & Schaa, D. *Heterogeneous Computing with OpenCL*. Morgan Kaufmann, Elsevier. ISBN 978-0-12-387766-6. First edition, 2012.

[9] Scarpino, M. *OpenCL in Action: how to accelerate graphics and computation*. Manning, 2012.

[10] Beyond3D. NVIDIA GT200 GPU and Architecture Analysis. http://www.beyond3d.com/content/reviews/51. June 2008.

[11] Fang, J., Varbanescu, A.L. & Sips, H. A Comprehensive Performance Comparison of CUDA and OpenCL. 2011 *International Conference on Parallel Processing (ICPP)*. Pages 216-225. September, 2011.

[12] OpenCL 1.2 specification. Khronos Group. Revision 19, November, 2012.

[13] van de Vijver, Y.A.J.R. *Ondersteuning NLR bij validatieproces GEVERS*. MN-GEV-16. Dutch National Aerospace Laboratory. October, 2007.

[14] van de Vijver, Y.A.J.R. & Guravage, M.A. *Validatie van TRIPAC, de nieuwe software voor analyse van externe veiligheid rond luchthavens*. NLR-TR-2003-651. Dutch National Aerospace Laboratory. November, 2003.

[15] Erkamp, R. Master's Thesis. *Acceleration of calculation of Third Party Risk around an airport using OpenCL.* Vrij Universiteit Amsterdam, The Netherlands. April 2013.