GPU Optimization of Base Form 820 Collective Artificial Spouse Retirement Plan

Niklas Schalck Johansson, Nikolaj Aaes and Hildur Flemberg
{nsjo, niaa, hufl}@itu.dk

IT University of Copenhagen

Table of Contents

Abstract. This paper explores how utilizing CUDA C and general purpose graphic processing units can make the calculation of reserves for the Base Form 820 Collective Artificial Spouse Retirement Plan, faster.

In its current state this calculation take more than 20 minutes per customer if a decent approximation of the reserve is to be achieved. The parallelized solution presented in this paper makes it up to INDSÆT TAL times faster than the original solution, meaning that the previous 20 minute execution now takes INDSÆT TAL seconds.

The solution enables insurance companies to estimate the reserves for Base Form 820 using relatively cheap hardware, much faster than previously.

Keywords: CUDA, parallelism, threads, blocks, insurance, capital, software, programming languages, Runge Kutta 4th order, death benefit.

1 Introduction

This report is written at the IT University of Copenhagen (ITU) in the spring term of 2013 in connection with a CUDA project supervised by Peter Sestoft from ITU and Hans Henrik Brandenborg Sørensen from the Technical University of Denmark (DTU). The report is addressed to people interested in exploiting the benefits of using general purpose graphical processing units (GPGPUs)[1] to optimize feasible computations in general, but specifically to people interested in CUDA.

In this project, we take a problem from the insurance industry that requires a significant amount of time to solve and show how to speed it up using the GPU. The problem concerns life insurance policies for married people, where one of them receives an amount of money some time after the other person has died. The goal, from the view of the insurance company, is to estimate its holding as accurately as possible to always be able to satisfy the obligation to the policyholder. Such an estimate is largely dependent on a guess of when the policyholder is going to die which can be anytime between signing of the policy and some, possibly large, number of years ahead in time. Furthermore, the time of death is dependent on several variables like age, gender etc.

The insurance company has a mathematical model that unifies all these variables and can be solved by a computer but it takes a lot of time to do - even with modern CPUs. Moreover, insurance companies typically issue several thousands

[1]Graphical Processing Units (GPUs) are most commonly used to render graphics in computer games. GPGPUs are "general purpose" variants that share the same architecture as GPUs but are optimized for computing purposes out of line with the graphics category. Since this project is not about graphics, we will refer to GPGPUs simply as GPUs, for shortness, throughout the rest of the report.

of policies, each of which has an impact on the complexity of the computation as well as the size of the overall reserves the company needs to hold back in case a disbursement is claimed. All these facts together make computing power a valuable entity. The goal of this project is to analyze the opportunity to speed up the computation for a single policyholder through parallelism with CUDA, i.e. to optimize the time it takes to compute the size of the reserves needed at any time during the life of a policyholder such that the insurance company can fulfill its obligations when the person dies.

The reader of the report is assumed to know the basic principles of concurrency and parallelism. No CUDA specific knowledge is required.

2 Background

- Related work, projects that precedes this project, why do we use the technology described, what is this new technology - Nævn at vi har fået udleveret matematik beforehand

2.1 Problem definition

How can CUDA C be utilized to optimize the calculation of the baseform 820 collective artificial spouse retirement plan using the Runge Kutta 4th order method?

2.2 Scope

The scope of this project is to handle a single customer at a time. We do not explore the possibilities for optimizing the process of using the algorithm on several customers at a time. Neither do we explore alternative methods for approximating differential equations.

2.3 Assumptions

This project was developed under several assumptions to limit the size of it. We assumed that the spouse of an insurance holder was always of the opposite gender. It would now be possible to take same-gender couples into account but it would require som modification to the code. We assumed that the function to determine the interest rate always returns 5 %.

3 The Math

The algorithm in this project is used to determine the lump sum of money the insurance company needs to possess to be able to pay the insurance holder's spouse in the case of his or her death. The payment to the spouse will be in the form of a life interest which are disbursed from the time of death of the insurance

holder unless the death occurs before the pension age. In that case the money will be disbursed after a grace period determined by the insurance company at the time the insurance was taken out.

We assume that the spouse of the insurance holder is of the opposite gender. If the insurance holder does not have a spouse at the time of his or her death the insurance is forfeited.

The algorithm used here is a 4th order Runge-Kutta solution with a fixed step-size (indsæt reference) where we use a series of constants determined before any calculation begins:

Constants

Name	**Meaning**
τ	The time of death of the insurance holder
r	The pension age
g	The grace period
x	The age of the insurance holder at calculation time ($t = 0$)
t	The time of calculation
h	The Stepsize of the Runge-Kutta solution

τ is expressed as $x + t$. Apart from this there is also a constant k which are determined by τ, r and g in the following manner:

How k is defined

If this statement holds	**then k equals**
$\tau < r$	g
$r \leq \tau < r + g$	$r + g - \tau$
$r + g \leq \tau$	0

The algorithm can be described as a combination of three models; an outer model that describes the life/death state of the insurance holder, a middle model that describes the married/unmarried state of the insurance holder and an inner model that describes the life/death state of the potential spouse.

NOTE: The description in the three model sections is loosely based on a describtion for Edlund[1] and some passages are directly translated from this document.

3.1 Outer model

The outer model is expressed as the following equation:

$$\frac{d}{dt}f(t) = r(t)f(t) - \mu_t(x + t)(S^d_{x+t}(t) - f(t))$$

Where $S^d_\tau$is the death benefit that for a τ year old at the time t is needed to cover the payment from the insurance company, $\mu_t(x + t)$ is the mortality rate

for a $(x+t)$ year old and $r(t)]$ is the interest rate function. In this project the interest rate function returns the constant 0.05.

The differential equation is solved from $t = 120-x$ to $t = 0$ with the boundary condition $f(120-x) = 0$

3.2 Middle model

The middle model is used to calculate $S^d_{x+t}(t)$ and is expressed with the following equation:

$$S^d_\tau(t) = \begin{cases} g_\tau \int f(\eta|\tau) a^I_{[\eta]+g}(t) d\eta & \tau \leq r \\ g_\tau \int f(\eta|\tau) a^I_{[\eta]+r+g-\tau}(t) d\eta & r \leq \tau \leq r+g \\ g_\tau \int f(\eta|\tau) a^I_{[\eta]}(t) d\eta & r+g \leq \tau \end{cases}$$

Where g_τ is the propability that a τ year is married and $f(\eta|\tau)$ is the probability distribution for, that a τ year old is married to a η year provided that the τ year old is married. This equation can be rewritten to this form:

$$\tfrac{d}{dn} f(\eta) = -g_\tau f(\eta|\tau) a^I_{[\eta]+k}(t)$$

here the DET-DER-TAL-SOM-LIGNER-EN-EKSPONENT-MEN-ER-NEDE-I-STEDET-FOR-OPPE for a is rewritten to $[\eta] + k$ where k can fall into three different categories as shown in THE-TABLE-CONSTANTS.

This differential equation is solved from $\eta = 120$ to $\eta = 1$ with the boundary condition $f(120) = 0$

3.3 Inner model

The inner model is used to calculate $a^I_{[\eta]+k}(t)$ and is expressed with the following equation:

$$\tfrac{d}{ds} f(s) = r(t+s) f(s) - 1_{s \geq k} - \mu_{t+s}(\eta+s)(0 - f(s))$$

Where t, η and k are constants and $\mu_{t+s}(\eta+s)$ is the mortality rate for a $(\eta+s)$ year old.

This differential equation is solved from $s = 120-\eta$ to $s = 0$ with the boundary condition $f(120-\eta) = 0$

3.4 Runge-Kutta 4th order

The Runge-Kutta method is a method for approximating differential equations that builds on Eulers method (INDSÆT REFERENCE) and the midpoint method (INDSÆT REFERENCE). When given a start point and a differential equation one can choose a stepsize and approximate the graph. When given a point (x_n, y_n), a differential equation f and a stepsize h one can use the Runge-Kutta

method to approximate the next point in the following way:

$k1 = hf(x_n, y_n)$
$k2 = hf(x_n + \frac{h}{2}, y_n + \frac{k1}{h})$
$k3 = hf(x_n + \frac{h}{2}, y_n + \frac{k2}{h})$
$k4 = hf(x_n + h, y_n + k3)$

$x_{n+1} = x_n + h$
$y_{n+1} = y_n + \frac{k1}{6} + \frac{k2}{3} + \frac{k3}{3} + \frac{k4}{6} + O(h^5)$ HVORFOR KAN VI IGNORERE $O(h^5)$??

3.5 Description of execution

To provide a better overview of how the math is applied we go through how an execution is performed. Before the execution begins certain value are give before hand, namely g, r and x. Additionally we need to decide on a stepsize before the execution can begin.

The first step in the execution is to use the outer model with a starting point. Since the outer model is to be solved for $t = 120 - x$ to $t = 0$ with the boundary condition $f(120 - x) = 0$ we can use the point $(120 - x, 0)$ as our starting point. The next step is to actually solve the outer model equation to calculate the next approximate point. All the components in the outer models differential equation can be computed using normal equations except $S^d_{x+t}(t)$. This component can be calculated using the middle model and when it computed the calculation of the next approximate point is completed. The outer model equation is solved four times for each approximation of the next point corresponding to calculating $k1$, $k2$, $k3$ and $k4$ in the Runge Kutta method. This can be repeated $(120 - x) \times stepsize$ times until the approximate point becomes $(0, y)$ which is the final point.

To calculate the missing component using the middle model we need to establish the value of k for the point. Additionally we are given the value t from the outer model. Like the outer model we need to use the middle model with at starting point. The middle model is to be solved for $\eta = 120$ to $\eta = 1$ with the boundary condition $f(120) = 0$ which means that we can use $(120, 0)$ as our starting point. As with the outer model all the components of the differential equation in the middle model can be computed using normal equations except $a^I_{[\eta]+k}(t)$. This can be calculated using the inner model which is the last model in the chain. When the inner model has computed a result the calculation of the next approximate point is completed. As in the previous model the middle model equation is calculated four times corresponding to calculating $k1$, $k2$, $k3$ and $k4$ in the Runge Kutta method. This can be repeated $119 \times stepsize$ times until the approximate point becomes $(0, y)$ which is the final point and the y-value of this point is returned to the outer model.

As the last part of the execution the inner model needs to calculate the last component used by the middle model. The inner model uses the t, k and η values provided by the middle model. This model uses a starting point in the same way as the other models. This model is to be solved for $s = 120 - \eta$ to $s = 0$ with the boundary condition $f(120-\eta) = 0$ which means we can use the point $(120-\eta, 0)$ as a starting point. Contrary to the other two models the inner model equation only contain components that can be computed using normal equations to find the next approximate point. For each approximation of a point the inner model equation is solved four times corresponding to calculating $k1$, $k2$, $k3$ and $k4$ in the Runge Kutta method. This can be repeated $120 - \eta \times stepsize$ times until the approximate point becomes $(0, y)$ which is the final point and the y-value of this point is returned to the middle model.

In any execution the outer model equation will be solved
$(120 - x) \times stepsize \times 4$ times,
the middle model equation will be solved
$((120 - x) \times stepsize) \times (119 \times stepsize \times 4)$ times
and the inner model equation will be solved
$((120 - x) \times stepsize \times 4) \times (\sum_{i=120\times stepsize}^{stepsize+1} ((120 - \frac{\eta}{stepsize}) \times stepsize \times 4) +$
$((120 - \frac{2\eta-1}{2\times stepsize}) \times stepsize \times 4) + ((120 - \frac{2\eta-1}{2\times stepsize}) \times stepsize \times 4) + (120 -$
$\frac{\eta-1}{stepsize}) \times stepsize \times 4))$ times.

For a normal execution with $stepsize = 4$, $g = 30$, $r = 80$ and $x = 115$ the outer model equation will be solved 80 times, the middle model equation will be solved 152.320 times and the inner model equation is solved 145.008.640 times.

4 Implementation

After we established the mathematical base we implemented a C solution. Instead of implementing the solution directly into CUDA C we chose to implement a C solution first to eliminate any bugs that were purely C specific. This also made it easier to debug since CUDA C program are generally harder to debug (INDSÆT REF?).

The first step was to implement the utility methods used for calculating trivial equations and were mostly taken directly from the original C# implementation. The methods in question are `gTau(double tau)`, `f(double eta, double tau)`, `k(double tau, double r, double g)`, `r_(double t)`, `GmFemale(double t)` and `GmMale(double t)`. These are all used to calculate components in each of the different differential equations.

5 CUDA

5.1 Architecture

Compute Unified Device Architecture (CUDA) is a parallel computing model created by Nvidia in 2006 based on a superset of the C programming language. In CUDA terminology, the computing system as a whole consists of a host (a traditional CPU) and one or more devices (GPUs) that are specially architectured to perform a massive amount of computations in parallel. Devices can execute kernel functions which is the CUDA term for a function that embeds code to be executed on the device.

Before a kernel execution is started, the host must make sure that all the data needed for the computation is available on the GPU. Conversely, the host is responsible for copying back the result data when the computation is completed. CUDA C exposes functions similar to the well known C functions *malloc* and free that allocate and free memory on the device similarly to how it is normally done on the hosting system. Additionally a function exists that performs the actual copying given source and target pointers. When all data is available on the device, the host can trigger the kernel function and start the computation.

When a device is assigned a kernel, the computing ressources of the device is partitioned into a number of streaming multiprocessors (SMP) that can each maintain a certain number of threads. The threads of an SMP are commonly known as a thread block. The total collection of blocks is known as a grid.

CUDA C extends the language with a whole new API with keywords and functions allowing the programmer to specify where a block of code should be executed, where memory should be allocated and so on. Furthermore, one is able to specify how the hardware ressources on the device should be configured to perform optimally while solving a particular problem, i.e. how many threads to use etc.

As the CUDA compiler uses the host compiler for C to compile traditional C code, any valid C program is accepted by the CUDA compiler, but the code will be executed entirely on the host. To execute kernel functions

As CUDA C is a superset of traditional C, any valid C program is accepted by the CUDA compiler, but the code is executed entirely on the host. To execute code on the GPU, one defines so called kernel functions which are ordinary C functions tagged with the "global" keyword.

6 Testing

To make sure that the C, C and CUDA programs produce similar results up to an acceptable (and adjustable) treshold, we run each program on a large data set and compare the outputs. Each of the 3 programs take 4 command line arguments which is (1) g - the number of years from the policyholder dies until the first rate is disbursed, (2) r - age of retirement, (3) x age at runtime and (4)

stepsize - the number of steps to take each year when the outer and the inner models are solved. Each program outputs a single number which is – roflcopter –.

We want all input from the test data set to fall within intervals that are chosen realistically and individually for each command line argument. It makes little sense to test the 3 programs with a retirement age of 200 since few people experience to become that old. Likewise, a big stepsize will make the overall test take very long time to complete which is impractical when testing during development. To avoid these problems, we made a program that generates test data in the shape of a plain text file. Each row in the file contains a comma separated list of numbers representing the 4 command line arguments to be fed to each of the 3 programs.

Next, we need a program that reads the lines of the test data file one by one, invokes the 3 programs in sequence and writes the corresponding results to a line in a new file. Obviously, this could be done by a single program as just described, but we wish to be able to test the C, C and CUDA programs separately during development without having to wait for the other 2 to finish, which might take longer time than the single program we want to test. For that reason, we make a test program for each of the 3 programs that reads a line from the test data file, invokes the program and appends the result to a text file that is private to the program being tested.

Finally, we have 3 text files with the results from each test run. We can then import the contents of each file into a column in a spread sheet and have the sheet compare the numbers on each row to identify matching and differing results.

The test scenario describe dabove was conducted on 3 machine with the following hardware specifications

Machine	The Malamanteau server	MacBook pro, Retina Mid 2012
OS	Windows 7 Pro 64-bit	OS X 10.8.3 (12D78)
CPU	Intel Xeon W3505 2.53 GHz	Intel Core i7 2,3 GHz
RAM	4 GB	8 GB 1600 MHz DDR3
GPU	Nvidia Tesla C2075[3]	Nvidia GeForce GT 650M [4]
CC	2.0	3.0
CUDA cores	448	384
Frequency of CUDA cores	1.15 GHz	Up to 900 MHz
Total dedicated memory	6 GB GDDR5	2 GB

For testing purposes we assume that the insurance holder is always a woman and the spouse is always a man. The individual test results kan be found in INDSÆT APPENDIX ELLER CD REF.

7 Benchmarks and Comparison

hvis vi deler steps op i blocks af 32 tråde kan der i worst case scenario være en sidste block med 1 aktiv og 31 inaktive tråde. skriv noget om memory bound og compute bound i rapporten - Benchmarking and comparison on speed between the current and new implementation

Tre slags grafer:
køretid og stepsize med konstante g,r,x. Tag ca 9 stikprøver.
CUDA køretid og rest tråde.
køretid og variable g,r,x.

8 Reflection / Discussion

improvements: Simpson løsning i middle? kig evt på en simpson integration når vi kigger på middle da det egentlig ikke er en differential ligning men et integrale

skriv evt noget om autotuning, hvor programmet først tester med forskellige mængder af tråde og blokke og vælger den hurtigste løsning.

Threats to validity??? kan det betale sig at køre dele af programmet på CPU'en?

- Did we achieve what we wanted? what did we discover during the project? What can be changed in future implementations?

When computing several customers one after another it could be beneficial to store the result of each calculation of the outer, middle and inner model and what parameters they were given. Whenever the program wanted to calculate value it could check if the calculation was already performed at a previous time and instantly get the result instead of calculating it again. If the result was not found in the storage it could calculate it itself and store the result afterwards for future use.

This approach is only beneficial when the lookup in the storage is faster than performing the calculation itself. It is also not beneficial if the amount of calculations that are identical, is not large enough. It would be interesting to examine the calculation collision with a large amount of executions with different customers. A drawback with this approach is that the storage can become very large over time and that could potentially lead to slower lookups. Since the approach stops being beneficial when the lookups are slower than the actual calculations it would be smart to examine when this occurs and evaluate what results is stored. If it occurs too fast one might consider to only store the middle or outer results because they are slower to calculate than the inner results.

9 Future work

This project has a limited scope and we would like to document suggestions for future expansion of the project:

Formal verification of correctness of code

We assume that the code produces the correct results because it is modeled after existing code and because our test of INDSÆT KORREKT TAL values are identical to the results produced by the existing code. Our test give a sense of correctness but without formal verification there is always the possibility that a corner case is not covered and will produce incorrect results.

Handling multiple customers

This project is built around the assumption that it would be run for a single customer at a time. If it is to have any real world use it must be able to run multiple customers at the same time. While this is possible with the current implementation is has not been tested or optimized. It would be interesting to examine how GPGPUs could be utilized when handling multiple customers and if the time used per customer could be even lower than it is when handling a single customer.

Variable interest curve

In this project the interest rate is assumed to be a constant of 5 %. However in the real world this is not the case. The interest rate is variable over time and the program should be able to take this into account.

Switch genders at runtime

When handling multiple customers in the same execution being able to switch the genders of the insurance holder and spouse at runtime is necessary. This is not a complex extension of the program but it is still something that is not implemented in this project. The difference between using female and male is the function used to calculate their mortality intensities. Changing whether the female or male function should be called for each customer and spouse should be trivial in the C and C# implementation but might have repercussions in the CUDA C implementation since it can generate branch divergence.

Same-gender marriages

As an addition to switching gender at runtime the program will be able to emulate the real world better if it supported same-gender marriages. The same considerations are present as the function to calculate mortality intensities is now the same for both the insurance holder and the spouse. Additionally the function used to calculate the probability that the insurance holder is married might be different for same-gender marriages.

10 Conclusion

Conclusion

11 Glossary

Insurance holder
The person who have made an agreement with the insurance company to enable his/her spouse to receive an amount of money in case of the his/her death.

Spouse
The person married to the insurance holder.

Life interest
A monthly payment to the spouse of the insurance holder

Death benefit
All the life interest payments added together. This is a construction created for the sake of calculation.

Grace period
The time period between the death of the insurance holder and the start of the payments.

Boundary condition
TODO

References

[1] EDLUND DOKUMENTET
[2] David B. Kirk, Wen-mei W. Hwu - Programming Massively Parallel Proccesors, A Hands-on Approach - Elsevier Inc. - 2010
[3] http://www.nvidia.com/docs/IO/43395/NV-DS-Tesla-C2075.pdf
[4] http://www.geforce.com/hardware/notebook-gpus/geforce-gt-650m/specifications