GPUs in Financial Computing Part III: ADI Solvers on GPUs with Application to Stochastic Volatility

Daniel Egloff
QuantAlea GmbH

Abstract
The first two articles in this series (Egloff, 2010a, 2010b) discussed the implementation and performance characteristics of a massively parallel PDE solver on GPUs for large collections of similar or related single-factor pricing problems. In the third article, we show how to implement efficient GPU solvers for two-factor models. Our implementation can price average size problems in the range of half a second and we beat an optimized CPU implementation by more than a factor of 70.

Keywords
partial differential equations, option pricing, finite-difference schemes, GPU, parallel cyclic reduction, Heston stochastic volatility.

1 Introduction
In the first two articles in this series (Egloff, 2010a, 2010b), we discussed the implementation and performance characteristics of a massively parallel PDE solver on GPUs, which we optimized to price a large collection of similar or related single-factor pricing problems with finite difference schemes. By pooling multiple pricing problems we could increase the level of data parallelism significantly and increase the overall utilization of the GPU. As a consequence, we achieved a speedup of a factor of 25 on a single GPU and up to a factor of 40 on a dual GPU configuration against an optimized CPU version.

In this article, we look at the design and implementation of efficient GPU solvers for two-factor models, with a focus on stochastic volatility models. We solve the resulting partial differential equations of two state variables with alternating direction implicit (ADI) schemes. We show that ADI-style schemes can be parallelized very efficiently on a GPU. Already a single pricing problem can utilize the full GPU capacity; a pooling of multiple pricing calculations to generate more parallelism is not necessary anymore.

In order to make our comparison as realistic as possible, we benchmark the GPU ADI solvers against optimized single-threaded and fully multi-threaded CPU implementations. On a recent C2050 Fermi GPU, we attain a speedup of more than a factor of 70 for a sufficiently large problem size.

2 Two Factor PDE Pricing
The partial differential equation to calculate the arbitrage-free price is a parabolic convection diffusion equation

$$\frac{\partial u}{\partial t} - \mathrm{L}_t u(t,x) = 0. \tag{1}$$

In this article, we consider two-factor models, such as baskets of two assets, where $x = (x_1, x_2)$ with x_i suitable transforms of the asset price, or stochastic volatility models, where $x = (s, v)$, with s the underlying asset price and v the unobservable stochastic volatility. A prominent example is the Heston stochastic volatility model introduced by Heston (1993), with dynamics

$$dS_t = S_t(r_t - q_t)dt + S_t\sqrt{v_t}\,dW_t^1, \tag{2}$$

$$dv_t = \kappa(\eta - v_t)dt + \sigma\sqrt{v_t}\,dW_t^2, \tag{3}$$

where $dW_t^1 dW_t^2 = \rho dt$ with correlation $\rho \in [-1,1]$. The model parameters are the initial volatility $v_0 > 0$, the mean reversion rate κ, long-run variance η, volatility of variance σ, and correlation ρ. The corresponding no-arbitrage PDE is given by

$$\begin{aligned}&\frac{\partial u}{\partial t} + \frac{1}{2}vs^2\frac{\partial^2 u}{\partial s^2} + \rho\sigma vs\frac{\partial^2 V}{\partial s\partial v} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V}{\partial v^2} + (r_t - q_t)s\frac{\partial V}{\partial s}\\ &+ (\kappa(\eta - v) - \Lambda(t,s,v))\frac{\partial V}{\partial v} - r_t V = 0\end{aligned} \tag{4}$$

where $\Lambda(t,s,v) = \lambda v$ is the (affine) market price of volatility risk and singles out a specific risk-neutral measure. Note that, without loss of generality, we can assume $\lambda = 0$.

For a parabolic PDE (1) on a two-dimensional states space, the implicit mixed-time Crank–Nicholson time discretization and finite-difference discretization of the spacial variables lead for every time step to a linear system of equations.[1] However, the matrices are no longer tridiagonal but instead have a bandwidth proportional to the minimum of the number of grid points along the coordinate axes of the two factors. The system can, in principle, be solved with LU factorization but this becomes ineffective for larger numbers of grid points.

Much more promising are alternating direction implicit (ADI) splitting schemes like the Douglas, Craig–Sneyd, or the more recent Hundsdorfer–

>

Verwer scheme, which are introduced and further discussed by Douglas (1962), Craig and Sneyd (1988), in 't Hout and Welfert (2007), and in 't Hout and Welfert (2009). For every time step, the ADI-style splitting schemes lead to many independent linear systems for every direction, which can be processed in parallel. For the commonly used finite-difference approximation of the spatial derivatives, these linear systems are essentially tridiagonal, up to a suitable permutation of the grid points, for which we can, again, use our fine-grained parallel tridiagonal solver from the previous articles.

We restrict our exposition to the particular case of the Heston stochastic volatility model, only briefly explain the ideas, and introduce the required notations to discuss the GPU implementation. Further details can be found in the paper by in 't Hout and Foulon (2010).

Let $[0, T]\times[s_{\min}, s_{\max}]\times[v_{\min}, v_{\max}]$ be a truncated domain and

$$\mathrm{T} = \{t_0,\dots,t_{n_t-1}\},\ \mathrm{S} = \{s_0,\dots,s_{n_s-1}\},\ \mathrm{V} = \{v_0,\dots,v_{n_v-1}\} \tag{5}$$

be suitable grids. Let $\mathbf{v}_n = (V(t_n, s_i, v_j),\ i = 0,\dots,n_s-1,\ j = 0,\dots,n_v-1)^{\mathrm{T}}$ be an approximation of the solution of (4) on the discretization nodes (5) at time t_n. We first approximate the spacial differential operator in (4) with finite differences and complement it with suitable boundary conditions, as described by in 't Hout and Foulon (2010). We can decompose the resulting linear operator L^n into a sum of three operators

$$\mathrm{L}^n = \mathrm{L}_0^n + \mathrm{L}_1^n + \mathrm{L}_2^n \tag{6}$$

where L_0^n is the part of L^n coming from the mixed-derivative term, L_1^n is the part that contains all the spacial derivatives, and L_2^n collects all the derivatives in direction of the variance coordinate v.

The zero-order term $r_1 V$ is evenly distributed to L_1^n and L_2^n. In contrast to in 't Hout and Foulon (2010), we apply finite-difference approximations, such that the resulting operators L_1^n and L_2^n become essentially tridiagonal up to a permutation of the grid points.

The Douglas scheme successively calculates the solution at time t_0 by

$$\mathbf{y}_0 = \mathbf{v}_{n+1} + \Delta_t^n \mathrm{L}^{n+1}\mathbf{v}_{n+1} \tag{7}$$

$$\mathbf{y}_j = \mathbf{y}_{j-1} + \Delta_t^n \theta(\mathrm{L}_j^n \mathbf{y}_j - \mathrm{L}_j^{n+1}\mathbf{v}_{n+1}), \quad j = 1,2 \tag{8}$$

$$\mathbf{v}_n = \mathbf{y}_2 \tag{9}$$

The forward Euler predictor step (7) is followed by two implicit but unidirectional corrector steps (8), which stabilize the predictor step. The Douglas scheme is a direct generalization of the classical ADI scheme for two-dimensional diffusion equations applied to the situation of a nonvanishing mixed spatial-derivative term. It is first-order-accurate, and only stable when applied to two-dimensional convection–diffusion equations with a mixed-derivative term if $\theta \geq \frac{1}{2}$. A more refined scheme is the Hundsdorfer–Verwer scheme given by

$$\mathbf{y}_0 = \mathbf{v}_{n+1} + \Delta_t^n \mathrm{L}^{n+1}\mathbf{v}_{n+1} \tag{10}$$

$$\mathbf{y}_0 = \mathbf{y}_{j-1} + \Delta_t^n \theta(\mathrm{L}_j^n \mathbf{y}_j - \mathrm{L}_j^{n+1}\mathbf{v}_{n+1}), \quad j = 1,2 \tag{11}$$

$$\tilde{\mathbf{y}}_0 = \mathbf{y}_0 + \frac{1}{2}\Delta_t^n(\mathrm{L}^n \mathbf{y}_2 - \mathrm{L}^{n+1}\mathbf{v}_{n+1}) \tag{12}$$

$$\tilde{\mathbf{y}}_0 = \tilde{\mathbf{y}}_{j-1} + \Delta_t^n \theta(\mathrm{L}_j^n \tilde{\mathbf{y}}_j - \mathrm{L}_j^n \mathbf{y}_2), \quad j = 1,2 \tag{13}$$

$$\mathbf{v}_n = \tilde{\mathbf{y}}_2 \tag{14}$$

The Hundsdorfer–Verwer scheme is an extension of the Douglas scheme, performing a second predictor step (12), followed by two unidirectional corrector steps (13). The advantage of the Hundsdorfer–Verwer scheme is that it attains order of consistency two for general operators L_0^n, L_1^n, L_2^n. Further alternatives are given by the Craig–Sneyd and its modified version, for which we refer to in 't Hout and Foulon (2010).

3 GPU Implementation

In contrast to single-factor models, the ADI schemes for two-factor models have sufficient parallelism for an effective GPU implementation of a single pricing problem; that is, there is no need to pool multiple pricing problems in order to increase the data parallelism.

We decompose the calculations for every time step into multiple kernel calls. The Douglas scheme requires three kernel calls per time step. The first kernel performs the explicit forward Euler predictor step (7) by exploiting the decomposition (6). Because the operators L_j, $j = 0,1,2$ are only used in matrix vector operations, we are free to use a discretization of the spatial derivatives, which does not necessarily lead to tridiagonal operators. This is particularly convenient for the mixed-derivative terms in L_0^n at the boundary, where we can use second-order forward- and backward-difference approximations. This kernel also calculates $\mathrm{L}_j^{n+1}\mathrm{v}_{n+1}$, which will be reused in the next two kernel calls to perform (8).

The second kernel calculates $\mathbf{y}_1$ in (8) for $j = 1$ by sweeping over the n_v slices $v = v_0,\dots,v = v_{n_v-1}$. In the inner nodes, the kernel solves a tridiagonal system, and at the face nodes $v = v_0$ and $v = v_{n_v-1}$ it uses the proper boundary conditions. Finally, the third kernel determines $\mathbf{y}_2$ from (8) for $j = 2$ by sweeping over the n_s slices $s = s_0,\dots,s = s_{n_s-1}$. For these two kernels, we must pay attention that the operators L_j^n, $j = 1,2$, which are used to solve the system of equations, are essentially tridiagonal, such that we can apply the parallel cyclic reduction solver or a variation thereof.

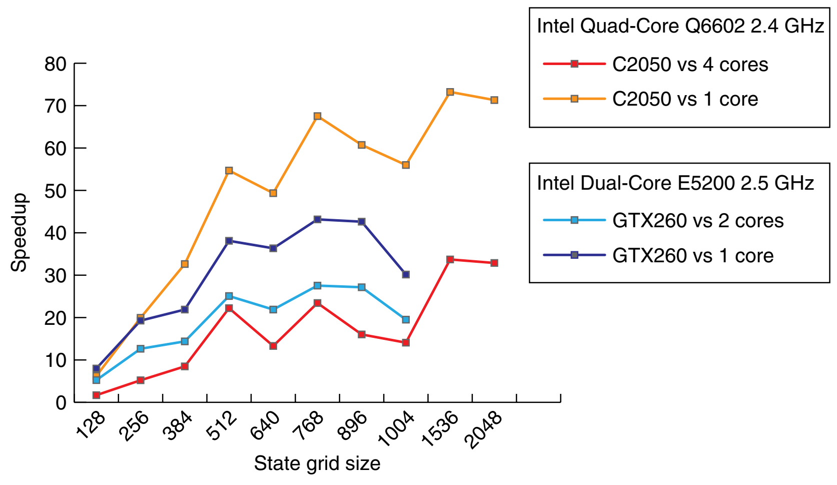
The suggested implementation of the Douglas scheme exhibits parallelism at two different levels. The first level is given by the sweeping over the slices in the two coordinate axes. The second level is inside of the solution algorithm of each tridiagonal system. The implementation of the Hundsorfer–Verwer scheme is very similar but slightly more complex because of the second predictor step and the following unidirectional corrector steps.

3.1 Performance results

We benchmark our GPU ADI solvers for the two-dimensional Heston stochastic volatility model against an optimized, fully multithreaded CPU implementation, for which we used the Intel threading building block library (Intel TBB Team 2010).

Figure 1 displays the relative performance figures for the Douglas scheme in single precision. On an Intel dual core E5200 2.5 GHz with a GTX260 GPU, the GPU ADI solver runs about 40 times faster than the CPU single-core version and 27 times faster than the optimized two-core version. On a Tesla C1060 or GTX260, the GPU ADI solver can handle state grids of at most 1,004 points because of shared memory and register limitations. The best speedup is achieved when the state grid size is near this limit. For small-scale problems, the speedup is not very large, due to the cost of allocating

Figure 1: Speedup ADI Douglas scheme on GTX260 versus dual core and C2050 versus quad core.



device memory. If the problem size is growing, the time required to allocate memory on the GPU becomes less dominant and the speedup increases significantly. The more accurate Hundsorfer–Verwer is about twice as intensive as the Douglas scheme, which is confirmed by our timings.

3.2 Fermi architecture

The new Fermi architecture is scaling up the compute capabilities of the 1.x generation GPUs by increasing the shared memory, number of registers, number of CUDA cores in a multiprocessor, etc. Algorithms designed for GPUs of compute capability 1.x can be adjusted to account for those changes in a relatively straightforward manner. However, there are other important architectural changes in Fermi, which must be exploited for performance optimization. The two most important changes are new cache memories, and, even more importantly, the latency to access registers and shared memory. With a GPU of compute capability 1.x, the access of registers and shared memory were comparable. In the new Fermi architecture, accessing data stored in registers is much faster. This provides motivation to place more data in registers and use them more carefully, which generally requires some changes in the code or algorithm.

The scaled-up hardware capabilities of the Fermi architecture allow us to process much larger problem sizes even faster (see Table 1). For example, on a new C2050 Fermi GPU, we can handle pricing problems with 3,056 state grid nodes in both dimensions. At these very large numbers of grid points, we get a speedup factor of more than 70 against a single-core implementation. The speed increase of the C2050 against a GTX260 GPU with compute capability 1.3 ranges from a factor of 1.6 up to a factor 2 for a state grid size of 896 and more. We mainly attribute this increase to faster register access in Fermi, as we designed our tridiagonal solver to place temporary data and intermediate results in registers whenever possible. Some smaller Fermi-specific optimizations could still be done when setting up the matrices for the Euler prediction steps. An interesting future optimization would be to increase the performance further by exploiting the new cache memory.

Table 1: Timing in seconds. Time to maturity = 2 years, time steps = 200 (Intel Quad-Core Q6602 2.4GHz, Fermi C2050, Cuda 3.1, Windows Vista).

Problem size (n_s, n_v)	One core	Four cores	GPU	Speedup single core	Speedup multicore
128	2.1	0.6	0.3	6.2	1.7
256	8.5	2.2	0.4	19.9	5.2
384	19.4	5.0	0.6	32.6	8.5
512	46.8	19.0	0.9	54.7	22.2
640	55.9	15.1	1.1	49.3	13.3
768	97.9	34.0	1.4	67.5	23.4
896	111.0	29.3	1.8	60.7	16.0
1004	133.6	33.6	2.4	56.0	14.1
1536	445.8	205.2	6.1	73.2	33.7
2048	793.3	365.7	11.1	71.3	32.9

Acknowledgment

The author would like to thank Gernot Ziegler and Tom Bradley for valuable feedback.

FOOTNOTE AND REFERENCES

1. As in our previous articles, we use suitably condensed and aligned grids, which leads to more complicated expressions for the finite difference discretization of derivatives but greatly improves the accuracy and stability of the numerical calculations.

Craig, I. J. D. and Sneyd, A. D. (1988). An alternating-direction implicit scheme for parabolic equations with mixed derivative terms. *Journal of Computational and Applied Mathematics* 16, 341–350.
Douglas, J. (1962). Alternating direction methods for three space variables, *Numerische Mathematik.* 4, 41–63.
Egloff, D. (2010a). GPUs in financial computing part I: High-performance tridiagonal solvers on GPUs. *Wilmott Magazine* September, 32–40.
Egloff, D. (2010b). GPUs in financial computing part II: Massively parallel PDE solvers on GPUs. *Wilmott Magazine* November, 50–53.
Heston, S. L. (1993). A closed-form solution for options with stochastic volatility with application to bond and currency options. *Review of Financial Studies* 9(2), 327–343.
in 't Hout, K. J. and Foulon, S. (2010). ADI finite difference schemes for option pricing in the Heston model with correlation. *International Journal of Numerical Analysis and Modeling* 7(2), 302–320.
in 't Hout, K. J. and Welfert, B. D. (2007). Stability of ADI schemes applied to convection diffusion equations with mixed derivative terms. *Applied Numerical Mathematics* 57, 19–35.
in 't Hout, K. J. and Welfert, B. D. (2009). Unconditional stability of second order ADI schemes applied to multi-dimensional diffusion equations with mixed derivative terms. *Applied Numerical Mathematics* 59, 677–692.
Intel TBB Team (2010). Intel® threading building blocks, Version 3.0. Available at www.threadingbuildingblocks.org/.