DELFT UNIVERSITY OF TECHNOLOGY

REPORT 07-12

Multi-asset option pricing using a parallel Fourier-based technique

C.C.W. Leentvaar and C.W. Oosterlee

ISSN 1389-6520

Reports of the Department of Applied Mathematical Analysis

Delft 2007

Multi-asset option pricing using a parallel Fourier-based technique

C.C.W. Leentvaar[1]*and C.W. Oosterlee[1,2]

[1]Delft University of Technology,
Numerical Analysis Group, c.c.w.leentvaar@tudelft.nl

[2]CWI, Center for Mathematics and Computer Science,
Amsterdam, the Netherlands, c.w.oosterlee@cwi.nl

October 5, 2007

Abstract

In this paper we present and evaluate a Fourier-based sparse grid method for pricing multi-asset options. This involves computing multi-dimensional integrals efficiently and we do it by the Fast Fourier Transform. We also propose and evaluate ways to deal with the curse of dimensionality by means of parallel partitioning of the Fourier transform and by incorporating a parallel sparse grids method. Finally, we test the presented method by solving pricing equations for options dependent on up to seven underlying assets.

1 Introduction

In this paper a parallel method (based on sparse grid and the FFT) for pricing multi-asset options is presented. The core of the method is the computation of a multi-dimensional integral. By means of the Fast Fourier Transform (FFT) we can compute this integral efficiently. Deterministic solvers (as opposed to Monte-Carlo methods) on tensor-product grids for multi-dimensional problems however suffer from the so-called *curse of dimensionality*, i.e., the exponential increase in the number of unknowns as the dimension d increases. This is also true for the FFT-based technique presented here. In order to reduce the complexity we combine a parallel partitioning method for the Fourier transform with a sparse grid method. The method is evaluated in terms of complexity analysis and by means of numerical experiments for multi-asset options for up to seven dimensions.

Besides others, FT-based methods have been successfully applied by Bakshi [1] and Scott [13] in computational finance. Here, we discuss a method by Lord et. al. [12], which has been developed in particular for pricing early-exercise options.

*The author wants to thank the Dutch Technology Foundation STW for financial support

This paper is organized in five sections. In Section 2, we describe the Fourier-based method for multi-asset options on tensor-product grids, called the multi-dimensional CONV method. Section 3 discusses the partitioning and corresponding parallelization of the method. This section also contains some parallel multi-asset results. In Section 4, we combine the method with the parallel sparse grid technique [17] which is based on [15] and present results from numerical experiments on sparse grids. In Section 5, we draw our conclusions from this work.

The options considered in this paper have their payoff given by:

- $\left(\prod_{j=1}^{d} S_j^{\frac{1}{d}} - K\right)^+$ (call on the geometric average of the assets).
- $\left(\sum_{j=1}^{d} c_j S_j - K\right)^+$, with c_j the basket weights (basket call).
- Options on the minimum or maximum of the underlying assets, for example: $(K - \max_j S_j)^+$, put on the maximum of the underlying assets.

2 The multi-dimensional CONV method

The method presented falls in the category of transform methods. These methods are based on the risk-neutral valuation formula, for options on a single asset:

$$V(t, S(t)) = e^{-r(T-t)} \mathbb{E}\left[V(T, S(T))\right], \tag{1}$$

where V denotes the value of the option, r is the risk-free interest rate, t is the current time, T is the maturity date and S represents the price of the underlying. Interest rate, r, is assumed to be deterministic here. Equation (1) is an expectation and can be valuated using numerical integration provided that the probability density function is known. Equation (1) can be written as,

$$V(t, x(t)) = e^{-r(T-t)} \int_{K^*}^{\infty} \left(e^{x(T)} - e^{K^*}\right) f(x) dx, \tag{2}$$

with $K^* = \ln K$, $x(t) = \ln S(t)$ and $f(x)$ the probability density function. The value of $V(t, x(t))$ tends to $S(0)$ as K^* tends to $-\infty$ and hence the call price is not square integrable. Therefore, the payoff is multiplied by damping factor $\exp(\alpha K^*)$, with $\alpha > 0$. The computation of the Fourier transform of the option value, ψ, can be done by using the characteristic function, $\phi(\omega)$, as proposed by Carr and Madan [5]:

$$\begin{aligned} \psi(\omega) &= e^{-r(T-t)} \int_{-\infty}^{\infty} e^{i\omega K^*} \int_{K^*}^{\infty} e^{\alpha K^*} \left(e^{x} - e^{K^*}\right) f(x) dx dK^* \\ &= \frac{e^{-r(T-t)} \phi(\omega - (\alpha+1)i)}{\alpha^2 + \alpha - v^2 + i(2\alpha+1)v}, \end{aligned} \tag{3}$$

where the characteristic function of the underlying is defined by

$$\phi(u) = \mathbb{E}\left[e^{iu \ln S(T)}\right]. \tag{4}$$

For a European call an exact solution exists. To compute the call price, the inverse Fourier transform has to be computed:

$$V(t, x(t)) = \frac{e^{-\alpha K^*}}{2\pi} \int_{-\infty}^{\infty} e^{-i\omega K^*} \psi(\omega) d\omega. \tag{5}$$

Following the same steps for basket options, however, would need the characteristic function of a basket value, which is not known in general. For common baskets quite accurate approximations can be obtained by assuming that the basket itself is an asset following the same distribution as one of the underlying assets [7]. Here, we evaluate a method which does not rely on such an approximation.

Like all transform-based methods, the CONV method [12] is based on the risk-neutral valuation formula (1). In the multi-dimensional version we need to compute:

$$V(t, \mathbf{x}(t)) = e^{-r(T-t)} \int_{\mathbb{R}^d} V(T, \mathbf{y}) f(\mathbf{y}|\mathbf{x}) d\mathbf{y}, \tag{6}$$

where $\mathbf{x} = \ln \mathbf{S}(t)$ is a vector of the log-asset prices, $\mathbf{y} = \ln \mathbf{S}(T)$ and $f(\mathbf{y}|\mathbf{x})$ is the probability density function of the transition of $\mathbf{x}$ at time t to $\mathbf{y}$ at time T.

With the right to exercise at certain times, t_n, before the maturity date, T, the *Bermudan option price* is defined by:

$$V(t_n, \mathbf{x}(t_n)) = \max \left\{ E(t_n, \mathbf{x}(t_n)) \ , \ e^{-r(t_{n+1}-t_n)} \int_{\mathbb{R}^d} V(t_{n+1}, \mathbf{y}) f(\mathbf{y}|\mathbf{x}) d\mathbf{y} \right\}, \tag{7}$$

with $E(t_n, \mathbf{x}(t_n))$ the exercise payoff at t_n.

At each exercise date, t_n, valuation of (7) can be interpreted as a European-style contract with maturity time t_{n+1} and "initial" time t_n. For the derivation of the multi-asset CONV method, we can therefore focus on (6) and keep the notation as in (6).

The main premise of the CONV method is that the transition density function $f(\mathbf{y}|\mathbf{x})$ equals the density of the difference of $\mathbf{y}$ and $\mathbf{x}$:

$$f(\mathbf{y}|\mathbf{x}) = f(\mathbf{y} - \mathbf{x}). \tag{8}$$

This holds for several models, such as geometric Brownian motion and, more generally, Lévy processes, which have independent increments. Then with $\mathbf{z} = \mathbf{y} - \mathbf{x}$, we have:

$$V(t, \mathbf{x}) = e^{-r(T-t)} \int_{\mathbb{R}^d} V(T, \mathbf{x} + \mathbf{z}) f(\mathbf{z}) d\mathbf{z}, \tag{9}$$

which is a cross-correlation between V and f. The cross-correlation operator can be treated as a convolution operator [9] and therefore the method is called the CONV method in [12].

The valuation of equation (9) can be done numerically for known probability density functions. For several asset price models, including the Lévy processes, however, only the characteristic function is known. When transforming the cross-correlation operator to Fourier-space, we have to deal with a product of

the Fourier transform of the payoff and the Fourier transform of the probability density function, which is the characteristic function.

The Fourier transform of a function can however only be taken if the function is $\mathbf{L}_1-$integrable. This is typically not the case for multi-asset payoff functions but damping techniques [5] and [12] are not available for multi-asset options in general. As an example we try to integrate a payoff of a two-dimensional basket call, which is damped by $e^{\alpha_1 x_1+\alpha_2 x_2}$,

$$\begin{aligned}
&\int_{\mathbb{R}^2} e^{\alpha_1 x_1+\alpha_2 x_2} K\left(c_1 e^{x_1}+c_2 e^{x_2}-1\right)^+ dx_1 dx_2 \\
=&K\int_{-\infty}^{-\ln c_2}\int_{\ln(1-c_2 e^{x_2})-\ln c_1}^{\infty} e^{\alpha_1 x_1+\alpha_2 x_2}\left(c_1 e^{x_1}+c_2 e^{x_2}-1\right) dx_1 dx_2 \qquad (10)\\
+&K\int_{-\ln c_2}^{\infty}\int_{-\infty}^{\infty} e^{\alpha_1 x_1+\alpha_2 x_2}\left(c_1 e^{x_1}+c_2 e^{x_2}-1\right) dx_1 dx_2.
\end{aligned}$$

The second term in (10) is unbounded because of the integration over $\mathbb{R}$ for x_1. It is not possible to find a proper value for α_1 to bound this integral.

In the multi-dimensional CONV method the computation is done numerically and therefore the domain of integration has to be truncated. If the payoff itself is truncated to a certain bounded subset Ω_d, it is $\mathbf{L}_1$-integrable. We therefore redefine the payoff function as,

$$V(T,\mathbf{x}) = \begin{cases} V(T,\mathbf{x}) & \mathbf{x}\in\Omega_d\subset\mathbb{R}^d \\ \mathbf{0} & \mathbf{x}\notin\Omega_d. \end{cases} \qquad (11)$$

We now take the Fourier transform of equation (9):

$$\begin{aligned}
e^{r(T-t)}\mathcal{F}\{V(t,\mathbf{x})\}(\boldsymbol{\omega}) &= \int_{\mathbb{R}^d} e^{i\boldsymbol{\omega}\mathbf{x}}\left[\int_{\mathbb{R}^d} V(T,\mathbf{x}+\mathbf{z})f(\mathbf{z})d\mathbf{z}\right]d\mathbf{x} \\
&= \int_{\mathbb{R}^d}\int_{\mathbb{R}^d} e^{i\boldsymbol{\omega}(\mathbf{x}+\mathbf{z})}V(T,\mathbf{x}+\mathbf{z})f(\mathbf{z})e^{-i\boldsymbol{\omega}\mathbf{z}}d\mathbf{z}d\mathbf{x}.
\end{aligned} \qquad (12)$$

Here the multi-dimensional Fourier transform, and its inverse, are defined as:

$$\begin{aligned}
\mathcal{F}\{h(\mathbf{x})\}(\boldsymbol{\omega}) &= \int_{\mathbb{R}^d} e^{i\boldsymbol{\omega}\mathbf{x}}h(\mathbf{x})d\mathbf{x}, \\
\mathcal{F}^{inv}\{H(\boldsymbol{\omega})\}(\mathbf{x}) &= \frac{1}{(2\pi)^d}\int_{\mathbb{R}^d} e^{-i\boldsymbol{\omega}\mathbf{x}}H(\boldsymbol{\omega})d\boldsymbol{\omega},
\end{aligned}$$

with, for example,

$$\int_{\mathbb{R}^d} e^{i\boldsymbol{\omega}\mathbf{x}}h(\mathbf{x})d\mathbf{x} = \int_{-\infty}^{\infty}\ldots\int_{-\infty}^{\infty} e^{i\omega_1 x_1}\ldots e^{i\omega_d x_d}h(x_1,\ldots,x_d)dx_1\ldots dx_d.$$

Changing the order of integration in (12) and using $\mathbf{y}=\mathbf{x}+\mathbf{z}$, we find:

$$\begin{aligned}
e^{r(T-t)}\mathcal{F}\{V(t,\mathbf{x})\}(\boldsymbol{\omega}) &= \int_{\mathbb{R}^d}\int_{\mathbb{R}^d} e^{i\boldsymbol{\omega}\mathbf{y}}V(T,\mathbf{y})f(\mathbf{z})e^{-i\boldsymbol{\omega}\mathbf{z}}d\mathbf{y}d\mathbf{z} \\
&= \int_{\mathbb{R}^d} e^{i\boldsymbol{\omega}\mathbf{y}}V(T,\mathbf{y})d\mathbf{y}\int_{\mathbb{R}^d} e^{-i\boldsymbol{\omega}\mathbf{z}}f(\mathbf{z})d\mathbf{z} \qquad (13)\\
&= \mathcal{F}\{V(T,\mathbf{y})\}(\boldsymbol{\omega})\phi(-\boldsymbol{\omega}),
\end{aligned}$$

with ϕ the characteristic function. After taking the inverse Fourier transform, the option price is found to be:

$$V(t,\mathbf{x}) = e^{-r(T-t)}\mathcal{F}^{inv}\left\{\mathcal{F}\{V(T,\mathbf{y})\}\phi(-\boldsymbol{\omega})\right\}. \tag{14}$$

In this paper the asset prices are modeled as correlated log-normal distributions. The characteristic function can be computed via the probability density function,

$$f(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}|}}\exp\left(-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}\right)^T\boldsymbol{\Sigma}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}\right)\right),$$

where $\mu_j = \left(r-\delta_j-\frac{1}{2}\sigma_j^2\right)(T-t)$, $\boldsymbol{\Sigma}$ is the correlation matrix with $[\sigma]_{jk} = \rho_{jk}\sigma_j\sigma_k$ and $|\boldsymbol{\Sigma}|$ its determinant. Dividend-yields δ_j, volatilities σ_j and the correlation coefficients ρ_{jk} are assumed to stay constant. The characteristic function reads, by using equation (4),

$$\phi(\boldsymbol{\omega}) = \exp\left(i\sum_{j=1}^{d}\mu_k\omega_k - (T-t)\sum_{j=1}^{d}\sum_{k=1}^{d}\rho_{jk}\sigma_j\sigma_k\omega_j\omega_k\right). \tag{15}$$

Equation (14) can now be solved numerically with the help of multi-dimensional quadrature rules. A computation using the FFT requires equidistant grids for $\mathbf{x}$, $\mathbf{y}$ and $\boldsymbol{\omega}$:

$$x_{j,k_j} = x_{j,0} + k_j dx_j, \tag{16}$$
$$y_{j,k_j} = y_{j,0} + k_j dy_j, \tag{17}$$
$$\omega_{j,n_j} = \omega_{j,0} + n_j d\omega_j, \tag{18}$$

where the index j denotes the j-th coordinate, $j = 1,\ldots,d$, and $k_j \in [0, N_j - 1]$ and $dx_j = dy_j$. The Nyquist relation has to be fulfilled to avoid aliasing:

$$dx_j \cdot d\omega_j = \frac{2\pi}{N_j}. \tag{19}$$

We abbreviate $x_{j,k_j} = x_{k_j}$, subscripts $k_1,\ldots,k_d$ by $\mathbf{k}$, the transformed vector by

$$\mathcal{F}\{V(T,y_{k_1},\ldots,y_{k_d})\} = \widehat{V}(T,\omega_{n_1},\ldots,\omega_{n_d}) = \widehat{V}_{\mathbf{n}}.$$

and the repeated sum by

$$\sum_{k_1=0}^{N_1-1}\cdots\sum_{k_d=0}^{N_d-1} = \sum_{\mathbf{k}=\mathbf{0}}^{\mathbf{N-1}}$$

Approximating the inner integral of (14) by the trapezoidal rule gives:

$$\widehat{V}_{\mathbf{n}} := \int_{\mathbb{R}^d} V(T,\mathbf{y})e^{i\boldsymbol{\omega}\mathbf{y}}d\mathbf{y} \approx d\mathbf{Y}\sum_{\mathbf{k}=\mathbf{0}}^{\mathbf{N-1}} Z_{\mathbf{k}}V_{\mathbf{k}}\exp(i\omega_{n_1}y_{k_1} + \ldots + i\omega_{n_d}y_{k_d}), \tag{20}$$

with $d\mathbf{Y} = \prod_{j=1}^{d} dy_j$, $Z_{\mathbf{k}} = \prod_{j=1}^{d} R_j(k_j)$ and the trapezoidal weights,

$$R_j(k_j) = \begin{cases} \frac{1}{2} & k_j = 0 \vee k_j = N_j - 1 \\ 1 & \text{otherwise.} \end{cases}$$

The terms $\exp(i\omega_{n_j} y_{k_j}), j = 1 \dots d$, can be rewritten as,

$$\begin{aligned} \exp(i\omega_{n_j} y_{k_j}) &= \exp(i(\omega_{j,0} + n_j d\omega_j)(y_{j,0} + k_j dy_j)) \\ &= \exp(i\omega_{j,0} y_{j,0}) \exp(i(\omega_{j,0} k_j dy_j + y_{j,0} n_j d\omega_j) \exp(\frac{2\pi i n_j k_j}{N_j}). \end{aligned}$$

We choose $\omega_{j,0} = -\frac{1}{2} N_j d\omega_j$ and $y_{j,0} = -\frac{1}{2} N_j dy_j$, meaning that the grids are centered at the origin. Furthermore, we introduce the standard DFT notation, $W_{N_j} = \exp(-\frac{2\pi i}{N_j})$, and have:

$$\begin{aligned} \exp(i\omega_{n_j} y_{k_j}) &= \exp(i\omega_{j,0} y_{j,0}) \exp(i(\omega_{j,0} k_j dy_j + y_{j,0} n_j d\omega_j) W_{N_j}^{-n_j k_j} \\ &= \exp(\frac{2\pi i N_j}{4}) \exp(-\pi i k_j) \exp(-\pi i n_j) W_{N_j}^{-n_j k_j} \\ &= (-1)^{k_j} (-1)^{n_j + \frac{N_j}{2}} W_{N_j}^{-n_j k_j}. \end{aligned}$$

Equation (20) can be written as:

$$\widehat{V}_{\mathbf{n}} \approx d\mathbf{Y} \sum_{\mathbf{k}=\mathbf{0}}^{\mathbf{N-1}} G_{\mathbf{k}} V_{\mathbf{k}} \prod_{j=1}^{d} (-1)^{n_j + \frac{N_j}{2}} W_{N_j}^{-n_j k_j}, \tag{21}$$

with $G_{\mathbf{k}} = Z_{\mathbf{k}} \prod_{j=1}^{d} (-1)^{k_j}$. Recognizing that

$$\begin{aligned} \mathcal{D}_d \{f_{\mathbf{k}}\} &= \sum_{\mathbf{k}=\mathbf{0}}^{\mathbf{N-1}} f_{\mathbf{k}} \prod_{j=1}^{d} e^{\frac{2\pi i n_j k_j}{N_j}} = \sum_{\mathbf{k}=\mathbf{0}}^{\mathbf{N-1}} f_{\mathbf{k}} W_{\mathbf{N}}^{-\mathbf{nk}} \\ \mathcal{D}_d^{inv} \{F_{\mathbf{n}}\} &= \frac{1}{\prod_{j=1}^{d} N_j} \sum_{\mathbf{n}=\mathbf{0}}^{\mathbf{N-1}} F_{\mathbf{n}} \prod_{j=1}^{d} e^{\frac{-2\pi i n_j k_j}{N_j}} = \frac{1}{\prod_{j=1}^{d} N_j} \sum_{\mathbf{n}=\mathbf{0}}^{\mathbf{N-1}} F_{\mathbf{n}} W_{\mathbf{N}}^{\mathbf{nk}} \end{aligned}$$

are the discrete Fourier transform (DFT) and inverse Fourier transform, respectively, we have:

$$\widehat{V}_{\mathbf{n}} \approx d\mathbf{Y} \prod_{j=1}^{d} \left((-1)^{n_j + \frac{N_j}{2}} \right) \cdot \mathcal{D}_d \left[G_{\mathbf{k}} V_{\mathbf{k}} \right], \tag{22}$$

where $\mathcal{D}_d$ is the $d-$dimensional (or d-times repeated) DFT. The outer integral of equation (14) is treated by the left-hand rectangle rule in accordance with the error analysis [12]. So, we have,

$$\begin{aligned} V(t, \mathbf{x_m}) &= e^{-r(T-t)} \mathcal{F}^{inv} \left(\widehat{V}_{\mathbf{n}} \cdot \phi_{\mathbf{n}} \right) \approx \frac{e^{-r(T-t)}}{(2\pi)^d} \int_{\mathbb{R}^d} \widehat{V}_{\mathbf{n}} \phi_{\mathbf{n}} e^{-i\boldsymbol{\omega}\mathbf{x}} d\boldsymbol{\omega} \\ &= d\boldsymbol{\Omega} \frac{e^{-r(T-t)}}{(2\pi)^d} \sum_{\mathbf{n}=\mathbf{0}}^{\mathbf{N-1}} \widehat{V}_{\mathbf{n}} \phi_{\mathbf{n}} e^{-i\omega_{n_1} x_{m_1} - \dots - i\omega_{n_d} x_{m_d}} \end{aligned} \tag{23}$$

with $\phi_{\mathbf{n}} = \phi(-\omega_{n_1}, \ldots, -\omega_{n_d})$, and by using (19),

$$d\mathbf{\Omega} = \prod_{j=1}^{d} d\omega_j = \prod_{j=1}^{d} \frac{2\pi}{N_j dy_j} = \frac{(2\pi)^d}{N^d d\mathbf{Y}}. \tag{24}$$

Again we can replace,

$$e^{-i\omega_{n_j} x_{m_j}} = (-1)^{m_j} (-1)^{n_j + \frac{N_j}{2}} W_{N_j}^{n_j m_j}. \tag{25}$$

Combining (23), (24) and (25) gives:

$$V(t, \mathbf{x_m}) \approx \frac{e^{-r(T-t)}}{N^d d\mathbf{Y}} \sum_{\mathbf{n=0}}^{\mathbf{N-1}} \widehat{V}_{\mathbf{n}} \cdot \phi_{\mathbf{n}} . \prod_{j=1}^{d} (-1)^{m_j} (-1)^{n_j + \frac{N_j}{2}} W_{N_j}^{n_j m_j}. \tag{26}$$

We see that the products $(-1)^{n_j + \frac{1}{2}N_j}$ vanish when combining (26) and (21). The combination of the two discretized integrals leads to the *multi-dimensional CONV method*:

$$V(t, \mathbf{x_m}) = \frac{e^{-r(T-t)}}{(2\pi)^d} \prod_{j=1}^{d} (-1)^{m_j} \mathcal{D}_d^{inv} \left[\phi_{\mathbf{n}} \mathcal{D}_d \left\{ V_{\mathbf{k}} G_{\mathbf{k}} \right\} \right]. \tag{27}$$

3 Parallel partitioning

Equation (27) is set up for tensor-product multi-dimensional grids. For an increasing number of dimensions, however, the total number of points will increase exponentially. This so-called *curse of dimensionality* [2] renders many nice sequential algorithms useless. Even on state of the art sequential computers, the memory is not large enough to store vectors of size $N = \prod_{j=1}^{d} N_j$, for d sufficiently large. One of the possibilities to solve on finer grids is to partition the problem and solve the different parts separately in parallel.

The partitioning chosen here is based on the *down-sampling* method [9]. The basis of this method is a splitting of the input vector (in our case the Fourier transformed payoff) into two parts: one part containing the even and one containing the odd points. These two DFTs can be computed independently and their result can be added. A straightforward sequential implementation of the DFT of size N uses N^2 computations, whereas the partitioned version needs $\left(\frac{N}{2}\right)^2$ computations for each DFT plus one summation. This partitioning can be continued. It will converge to $O(N \log N)$ computations, like the FFT. Taking N as a power of two is the most efficient choice. Equation (27) is based on the transform of the payoff and the inverse transform of the product. This structure complicates the partitioning to some extent, but it can still be used, as described below.

Consider first the one-dimensional version of equation (27):

$$H_m = \sum_{n=0}^{N-1} \phi_n \widehat{V}_n W_N^{mn}, \tag{28}$$

where we omit the discounting factor. As mentioned, we split (28) into two sums of size $M = N/2$:

$$\begin{aligned} H_m &= \sum_{n=0}^{M-1} \phi_{2n} \widehat{V}_{2n} W_N^{2nm} + \sum_{n=0}^{M-1} \phi_{2n+1} \widehat{V}_{2n+1} W_N^{(2n+1)m} \\ &= \sum_{n=0}^{M-1} \phi_{2n} \widehat{V}_{2n} W_M^{nm} + W_N^m \sum_{n=0}^{M-1} \phi_{2n+1} \widehat{V}_{2n+1} W_M^{nm}, \end{aligned} \tag{29}$$

where we use

$$W_N^{2nm} = e^{-\frac{2\pi i 2nm}{N}} = e^{-\frac{2\pi imn}{M}} = W_M^{mn}.$$

The two DFTs in (29) can be solved in parallel. The inner parts $\phi_{2n}, \phi_{2n+1}, \widehat{V}_{2n}$ and $\widehat{V}_{2n+1}$ can be computed by addressing the appropriate points. $\widehat{V}_{2n}$ and $\widehat{V}_{2n+1}$ are based on a size N vector, $\widehat{V}$, which we partition as well.

For the even elements, we can write:

$$\begin{aligned} \widehat{V}_{2n} &= \sum_{k=0}^{N-1} G_k V_k W_N^{2nk}, \\ &= \sum_{k=0}^{M-1} G_k V_k W_M^{nk} + \sum_{k=M}^{N-1} G_k V_k W_M^{nk} \\ &= \sum_{k=0}^{M-1} G_k V_k W_M^{nk} + W_M^{nM} \sum_{k=0}^{M-1} G_{k+M} V_{k+M} W_M^{nk}, \end{aligned} \tag{30}$$

and for the odd elements:

$$\widehat{V}_{2n+1} = \sum_{k=0}^{M-1} G_k V_k W_N^k W_M^{nk} + W_N^M W_M^{nM} \sum_{k=0}^{M-1} G_{k+M} V_{k+M} W_N^k W_M^{nk}. \tag{31}$$

We observe that (30) and (31) are again a sum of two DFTs of size $N/2$. When the splittings (29), (30) and (31) are combined, we find the one-dimensional partitioned version of equation (28).

The multiple partitioned version is based on a repetition of this splitting which we derive for β parts, with β a power of two. The size of the computations is then $M = \beta^{-1}N$. The points used in the splitting of the inverse transform are given by $\beta n + q$, with $q \in [0, \beta - 1]$. So, the multiple partitioned version of equation (28) reads, using $W_N^{\beta nm} = W_N^{nm}$:

$$H_m = \sum_{q=0}^{\beta-1} \sum_{n=0}^{M-1} \phi_{\beta n+q} \widehat{V}_{\beta n+q} W_N^{m(\beta n+q)} = \sum_{q=0}^{\beta-1} W_N^{mq} \sum_{n=0}^{M-1} \phi_{\beta n+q} \widehat{V}_{\beta n+q} W_M^{mn}. \tag{32}$$

The partitioning into the odd and even parts can now be included:

$$\begin{aligned}\widehat{V}_{\beta n+q} &= \sum_{k=0}^{N-1} V_k G_k W_N^{-(\beta n+q)k} \\ &= \sum_{p=0}^{\beta-1}\sum_{k=0}^{M-1} V_{k+pM} G_{k+pM} W_N^{-(\beta n+q)(k+pM)} \\ &= \sum_{p=0}^{\beta-1} W_\beta^{-pq} \sum_{k=0}^{M-1} V_{k+pM} G_{k+pM} W_M^{-nk} W_N^{-qk},\end{aligned} \tag{33}$$

where we used

$$W_N^{-\beta npM} = e^{2\pi inp} = 1 \quad \text{and} \quad W_N^{-pqM} = W_\beta^{-pq}.$$

Combining (32) and (33), we obtain the one-dimensional multiple split version of equation (28):

$$\begin{aligned}H_m &= \sum_{q=0}^{\beta-1} W_N^{mq} \sum_{n=0}^{M-1} \phi_{\beta n+q} W_M^{mn} \sum_{p=0}^{\beta-1} W_\beta^{-pq} \sum_{k=0}^{M-1} V_{k+pM} G_{k+pM} W_M^{-nk} W_N^{-qk} \\ &= \sum_{p=0}^{\beta-1}\sum_{q=0}^{\beta-1} W_\beta^{-pq} W_N^{mq} \mathcal{D}^{inv}\left(\phi_{\beta n+q}\mathcal{D}\left[V_{k+pM} G_{k+pM} W_N^{-qk}\right]\right).\end{aligned} \tag{34}$$

This partitioning can be generalized to the multi-dimensional case. We then have a partitioning *vector* $\boldsymbol{\beta}$ containing β_j-parts for each coordinate j. The points in the outer transform (32) are represented by $\boldsymbol{\beta n}+\boldsymbol{q} = (\beta_1 n_1 + q_1, \ldots, \beta_d n_d + q_d)$. The points of the payoff addressed in (33) are represented by $\mathbf{k} + \mathbf{pM} = (k_1 + p_1 M_1, \ldots, k_d + p_d M_d)$. When using the multi-dimensional summation, the multiple partitioned version of equation (27) reads:

$$V(t,\mathbf{x}) = \frac{e^{-r(T-t)}}{(2\pi)^d} \sum_{\mathbf{p}=\mathbf{0}}^{\boldsymbol{\beta}-\mathbf{1}} \sum_{\mathbf{q}=\mathbf{0}}^{\boldsymbol{\beta}-\mathbf{1}} W_{\boldsymbol{\beta}}^{-\mathbf{pq}} W_{\mathbf{N}}^{\mathbf{mq}} \mathcal{D}_d^{inv}\left[\phi_{\boldsymbol{\beta}\mathbf{n}+\mathbf{q}} \mathcal{D}_d\left\{V_{\mathbf{k}+\mathbf{pM}} G_{\mathbf{k}+\mathbf{pM}} W_{\mathbf{N}}^{-\mathbf{qk}}\right\}\right]. \tag{35}$$

We see that if one of the coordinate grids is split into two parts, i.e. $\beta_k = 2, \beta_{j\neq k} = 1$, the computation of equation (35) would be a combination of 4 DFTs of size $\frac{N_k}{2}\prod_{j=1,j\neq k}^d N_j$. If the same coordinate would be partitioned again, we would deal with 16 DFTs of size $\frac{N_k}{4}\prod_{j=1,j\neq k}^d N_j$. The parallel efficiency is low in this case, as the number of processors needed grows quadratically with β_j. Therefore, we rewrite equation (35) as,

$$V(t,\mathbf{x}) = \frac{e^{-r(T-t)}}{(2\pi)^d} \sum_{\mathbf{q}=\mathbf{0}}^{\boldsymbol{\beta}-\mathbf{1}} W_{\mathbf{N}}^{\mathbf{mq}} \mathcal{D}_d^{inv}\left[\phi_{\boldsymbol{\beta}\mathbf{n}+\mathbf{q}} \mathcal{D}_d\left\{\sum_{\mathbf{p}=\mathbf{0}}^{\boldsymbol{\beta}-\mathbf{1}} V_{\mathbf{k}+\mathbf{pM}} G_{\mathbf{k}+\mathbf{pM}} W_{\mathbf{N}}^{-\mathbf{qk}} W_{\boldsymbol{\beta}}^{-\mathbf{pq}}\right\}\right]. \tag{36}$$

In this case the computations are partitioned over the $\mathbf{q}$-sum into $B = \prod_{j=1}^d \beta_j$ parts. Each processor now has to compute the $\mathbf{p}$ parts of the payoff function.

The summation over $\mathbf{p}$ could also be done in parallel with communication among the processors. A drawback of allowing communication for high-dimensional problems is the need to transfer very large vectors from one processor to another. We certainly need the communication when solving *early exercise* options in parallel, but not for European options. So, early exercise options would be parallelized efficiently on a parallel machine with some form of shared memory. An alternative to this type of parallelization could be the sparse grid method for which parallelization is straightforward. We focus on the communication-less version of this parallel approach and solve European style options with it.

3.1 Complexity analysis

We now evaluate the parallelization technique to solve equation (27) by a complexity analysis, and first briefly summarize the procedure to solve (27) below.

1. Compute the payoff on the tensor-product grid;
2. Multiply the payoff by the function $G_{\mathbf{k}}$;
3. Take the fast Fourier transform;
4. Multiply the result by the characteristic function;
5. Take the inverse fast Fourier transform of the product;
6. Multiply it by the discount factor.
7. For Bermudan options: Take the maximum of this value and the payoff function at t_n. Repeat the procedure from step 2 until t_0 is reached.

The construction of the payoff function is a significantly faster procedure than the computation of the two FFTs and the multiplication by the characteristic function. We distinguish three portions of time consumption during the solution process of European options:

- T_{pay} is the time needed to construct the payoff including the multiplication with the function $Z_{\mathbf{k}}$ and $W_{\mathbf{N}}^{-\mathbf{qk}}$ in (36),
- T_{four} is the time in steps 3 to 6 in the algorithm,
- T_{add} is the additional time needed for starting the computation, reading and writing files.

We assume here that T_{add} is negligible. The total time needed to compute equation (27) is then $T_{tot} \approx T_{pay} + T_{four}$. We further assume that $T_{four} = AT_{pay}$. If technique (36) is used and we partition the problem in B parts, the computational time per processor reads,

$$T_{tot,split} = T_{pay} + \frac{1}{B} T_{four} = \frac{A+B}{B} T_{pay}, \tag{37}$$

with $B = \prod_{j=1}^{d} \beta_j$. If there are Q identical processors available, then the parts B can be distributed over the Q processors. Ideally Q is a divisor of B. The number of parallel processes is therefore equal to $\lceil \frac{B}{Q} \rceil$ and the computational time reads:

$$T_{tot,split} = \left\lceil \frac{B}{Q} \right\rceil \frac{A+B}{B} T_{pay} \tag{38}$$

In our applications, typically, $A \in [4, 12]$ for $B = 1$.

3.2 Full grid experiments

We evaluate the CPU-times of the parallel CONV method for some multi-dimensional experiments on tensor-product grids. We first evaluate the option on the geometric average for which we compare the numerical result with an analytic solution [14]. This solution can be obtained by the use of a coordinate transformation $y = \prod_{j=1}^{d} e^{\frac{x_j}{d}}$. The option price can then be obtained via the one-dimensional Black-Scholes formula with:

$$\widehat{\sigma} = \sqrt{\frac{1}{d^2} \sum_{j=1}^{d} \sum_{K=1}^{d} \rho_{jk} \sigma_j \sigma_k}, \quad \widehat{\delta} = \frac{1}{d} \sum_{j=1}^{d} \left(\delta_i + \frac{1}{2} \sigma_i^2 \right) - \frac{1}{2} \widehat{\sigma}^2.$$

In Table 1, the prices for the 4D call option on the geometric average of the assets are presented for a different number of grid points. The first column in Table 1 represents the number of grid points per coordinate $N_j = 2^{n_f}$, $j = 1, \ldots, 4$. The final computation, $n_f = 7$, requires 4 GB of memory and has a complexity of 2^{28}-points.

The option parameters chosen are $r = 6\%$, $\sigma_j = 0.2$, $\delta_j = 0.04$, $\rho_{jk} = 0.25$ if $j \neq k$ and $T = 1$. The strike price is €40, as is $\mathbf{S}(0)$.

The desired accuracy of errors being less than €0.01 is achieved for $n_f = 5$. We observe a second order convergence on the finer grids. The right-side part of Table 1 presents timings on a parallel machine, which consists of nodes with two processors each, having 8 GB of memory. Parameter A, as in (37) is also given.

$d = 4$	Call on the geometric average			CPU times with eq. (36)				
n_f	Price	Error	Ratio	B=1	B=2	B=4	B=16	A
3	1.962	2.0×10^{-1}	5.3	<0.1	<0.1	<0.1	<0.1	
4	2.128	3.8×10^{-2}	5.4	<0.1	<0.1	<0.1	<0.1	
5	2.156	9.3×10^{-3}	4.1	0.5	0.2	0.1	<0.1	4.5
6	2.163	2.3×10^{-3}	4.0	9.4	4.9	3.0	1.6	6.2
7	2.165	5.8×10^{-4}	4.0	164.1	85.1	45.2	25.2	7.1

Table 1: Option prices for the 4D geometric average call option with the parallel timings (in sec.). Last column gives A (37).

Based on these results we conclude that the partitioning strategy reduces total CPU time well. Parallel efficiency would improve on finer grids and in higher dimensions.

We now consider problems that require more than the maximum available physical memory per processor. The parallel partitioning is then mandatory. In Table 2, we present the prices of a digital put on the geometric average of five assets. As the payoff function of the digital option (it will pay an amount of €1 when the geometric average is less than the strike price in our experiment) has a discontinuity along the hyper-surface $\prod_{j=1}^{d} e^{\frac{x_j}{d}} = 1$, it is expected that this leads to only first order error convergence. Table 2 indeed displays first order convergence (again the exact solution is known for the digital put on the geometric average) and we see that a grid size with $n_f = 6$ is not sufficient to reach the desired accuracy.

Table 3 presents the solution of a 6D standard basket put with equally weighted assets ($c_i = \frac{1}{6}$). The error convergence is irregular for this payoff, but at least of second order. The size of $n_f = 5$ is again sufficient to reach the desired accuracy. The CPU times for $B = 32$ in Tables 2 and 3 are estimated times when the number of processors Q is equal to the number of parts B.

$d = 5$	Digital put on the geometric average				CPU times	
n_f	Price	Error	Ratio	B=4	B=32	A
2	0.81	3.36×10^{-1}	1.49	<0.1	<0.1	
3	0.32	1.49×10^{-1}	2.26	<0.1	<0.1	
4	0.40	7.43×10^{-2}	2.00	0.2	0.1	4.0
5	0.43	3.71×10^{-2}	2.00	1.8	1.1	4.5
6	0.45	1.86×10^{-2}	2.00	295.6	91.1	8.7

Table 2: Option prices for the 5D geometric average digital put, plus parallel timing results and parameter A from (37).

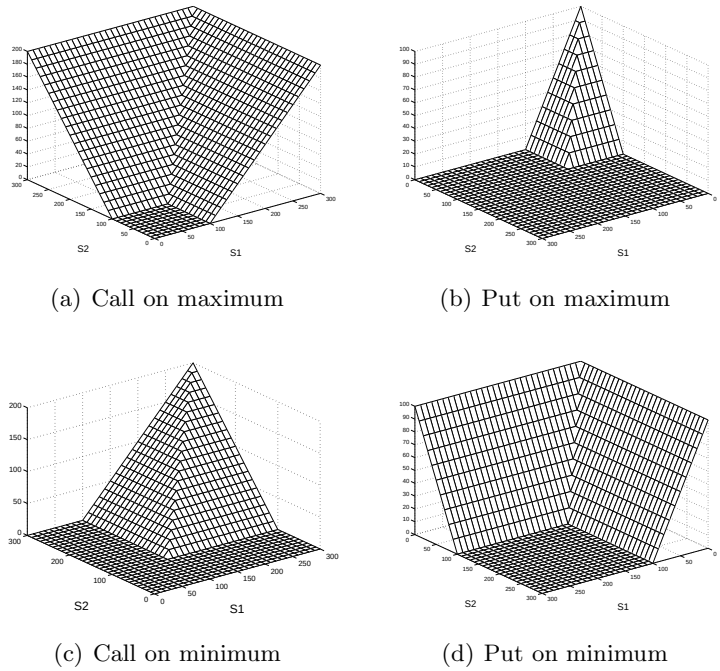
$d = 6$	Basket put			CPU times	
n_f	Price	Error	Ratio	B=4	B=32
2	1.26	1.25		<0.01	<0.01
3	1.52	2.63×10^{-1}	4.7	0.09	<0.01
4	1.51	1.70×10^{-2}	15.5	5.02	1.2
5	1.50	2.62×10^{-3}	6.5	334.34	111.1

Table 3: Option prices for the 6D basket put, plus parallel timing results.

The hedge parameters can easily be obtained using the CONV method. We refer to the appendix for the derivation and some numerical results for Delta.

In the next section, we will describe the sparse grid technique to solve multi-asset options. The sparse grids technique can be chosen if the required memory or the required number of processors for the partitioned full grid version is just too large. However, the efficient use of the sparse grid technique in computational finance is *seriously restricted* by the types of multi-asset option contracts in use. An acceptable accuracy with the sparse grids method can only be expected if the solution has bounded mixed derivatives. The payoffs of the

examples presented in Tables 1, 2 and 3 do not have this property. It may be possible to transform a payoff so that the kink (or discontinuity for a digital option) is *aligned* with a grid line [11], but this cannot be done for every payoff. A call or put option based on the *maximum or minimum* of the underlying assets has its non-differentiability on grid lines, see Figure 1. It is therefore



(a) Call on maximum (b) Put on maximum

(c) Call on minimum (d) Put on minimum

Figure 1: Payoff for 2D options on the maximum or minimum of assets.

expected that these options can be handled well in the sparse grid setting.

4 Sparse grids

The partitioning of equation (27) to get (36) is not sufficient to deal with the curse of dimensionality. It helps to get problems of moderate size into the memory or to speed up the computation of medium-sized problems. A 5D problem with 2^6 grid points per coordinate would need vectors containing 2^{30} elements (about 16 GB of memory). With a maximum of 4 GB of memory per processor, we could solve the problem on 4 processors (splitting the problem twice). However, a 7D problem with 2^7 grid points per coordinate is too large to partition with a moderate number of processors. Vectors on this grid will consist of 2^{49} elements. If machines of 4 GB memory are available, the number of splittings required is 21 and we would have 2^{21} subproblems of a reasonable size. Of course, several subproblems can be solved on a single processor as the FFT-based computation is typically fast. With 2^{11} subproblems on one processor, we would need $2^{10} \approx 1000$ processors. The (computer dependent) restrictions on the number of dimensions and the tensor-product grid sizes that can be efficiently handled in parallel can be calculated easily.

The sparse grid method, developed by Zenger and co-workers [4, 17] is an approach which allows the solution of problems with higher dimensionality. The solution is obtained via a combination of approximations obtained on relatively coarse high-dimensional grids, that are called subproblems here. The solution of this combination technique *mimics* the solution on a full tensor-product grid.

Let's consider a full grid with $N_j = 2^{n_s}$, i.e., the number of grid points for coordinate direction j. The sparse grid method combines subproblems with a maximum number of 2^{n_s}-points in one coordinate. The other coordinates of the subproblem are discretized with a minimum number of grid points per direction, 2^b, called 'the base' here. The number of subproblems to combine depends on the number of dimensions, the base, and the number of points of the full grid to mimic. These subproblems are ordered in d layers with the complexity of each subproblem within the layer being the same. The ordering of layers in the sparse grid combination technique is defined via a *multi-index*.

Definition 1 *A multi-index $\mathcal{I}_{d,\ell}$ related to a $d-$dimensional grid is the collection of the sets of numbers $(n_1, n_2, \ldots, n_d)$ which have the property:*

1. $n_j \geq b, \ \forall j,$

2. $\sum_{j=1}^{d} n_j = \ell$ *and* $\ell = n_s - j + (d-1)b + 1$ *for* $j = 1, \ldots, d$

An element from the multi-index is called a subproblem-index. The size of the subproblem is $(2^{n_1} \times 2^{n_2} \times \ldots \times 2^{n_d})$. The multi-index is empty if one of the two conditions does not hold.

The combination of the subproblem solutions, by means of interpolation, at a certain layer ℓ, is just the sum of these solutions on the grids from the multi-index $\mathcal{I}_{d,\ell}$. The first layer of subproblems consists of grids of size $(2^{n_s} \times 2^b \times \ldots \times 2^b)$. The *layer number* for this grid is $\ell = n_s + (d-1)b$. Grids of size $(2^b \times 2^b \times \ldots \times 2^{n_s})$ and $(2^{b+2} \times 2^b \times \ldots \times 2^{n_s-2})$ are, for example, also present in this layer. The next layer is of size $\ell = n_s - 1 + (d-1)b$. So, the d layers needed in the sparse grid combination technique are,

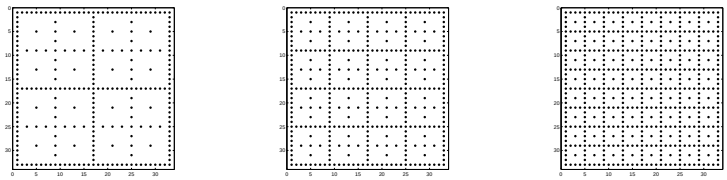
$$\ell = n_s - j + (d-1)b + 1 \quad j = 1 \ldots d. \tag{39}$$

To form the solution a weighted combination of layers is taken with the binomial coefficients as the weights [4],

$$V_{combined} = \sum_{j=1}^{d} (-1)^{j+1} \binom{d-1}{j-1} \sum_{q \in \mathcal{I}_{d,\ell}} V_q, \ (\ell = n_s - j + (d-1)b + 1). \tag{40}$$

This construction is similar to the difference quadrature formulation of Smolyak [15]. The combined solution leads to a *sparse grid solution*, see Figure 2.

Example: Consider a 3D grid of size 16^3, whose solution we would like to mimic by the sparse grid method. Now $n_s = 4$, $d = 3$ and we have 3 layers of subproblems. Suppose that we need a grid that consists of at least 4 points per coordinate ($b = 2$), then we find:



(a) b=1 (b) b=2 (c) b=3

Figure 2: Construction of a 2D sparse grid: combined solution

- First layer, $\ell = 8$, with grids $(16 \times 4 \times 4), (8 \times 8 \times 4), (4 \times 16 \times 4), (8 \times 4 \times 8), (4 \times 8 \times 8)$ and $(4 \times 4 \times 16)$.
- Second layer, $\ell = 7$, with grids $(8 \times 4 \times 4), (4 \times 8 \times 4)$ and $(4 \times 4 \times 8)$.
- Third layer, $\ell = 6$, with grid $(4 \times 4 \times 4)$.

If the base would be 8 ($b = 3$), then we would have two layers (as a third layer would violate Condition 2 in Definition 1 and therefore give an empty multi-index),

- First layer, $\ell = 10$, with grids $(16 \times 8 \times 8), (8 \times 16 \times 8)$ and $(8 \times 8 \times 16)$.
- Second layer, $\ell = 9$, with grid $(8 \times 8 \times 8)$.

In this paper, the grid is chosen such that the value of the origin is the grid's central point. This origin is then a grid point of every subproblem. The solution at another grid point can either be found by shifting the grid so that the point lies in the center, or we can use interpolation to find the value.

Although we do not have an exponentially growing number of points with this method, number of subproblems to be solved for increasing dimensions increases significantly. It is related to the number of elements, $\mathcal{M}_{d,\ell}$, in the multi-index, which is based on layer number, ℓ, in equation (39). For a given n_s, b and d it reads:

$$\mathcal{M}_{d,\ell} = \binom{\ell - d(b-1) - 1}{d-1}. \tag{41}$$

The total number of points on this layer reads, $\mathcal{N}_\ell = 2^{\ell+(d-1)} \cdot \mathcal{M}_{d,\ell}$, and the total number of points in a sparse grid computation, mimicking a full grid of size 2^{n_s} in each direction, follows from (39), (40), and (41):

$$\begin{aligned} \mathcal{N}_{total} &= \sum_{j=1}^{d} 2^{n_s - j + 1 + (d-1)b} \binom{n_s - j + 1 + (d-1)b - d(b-1) - 1}{d-1} \\ &= 2^{n_s - d + 1 + (d-1)b} \sum_{j=1}^{d} 2^{d-j} \binom{n_s - j - b + d}{d-1}. \end{aligned} \tag{42}$$

Example: We return to the example of a 7D problem with $n_s = 7$, and choose $b = 2$. From the definition of the multi-index, it follows that 5 layers

are needed, because two layers have empty multi-indices. The ℓ-values range from $15 \leqslant \ell \leqslant 19$, the complexities from 2^{21} points on the lowest, $\ell = 15$, to 2^{25} on the top layer ($\ell = 19$). The number of subproblems is 330, ranging from 1 for the lowest to 210 for the top layer. The overall grid complexity according to (42) is 2^{33}-points. The subproblems can be solved in parallel and we would not need the additional FFT parallelization, as 2^{25}-points fit in the memory of a 4GB machine. The parallel efficiency of the sparse grid technique is high, because of the significantly large number of subproblems. All subproblems can be distributed, for example, to only two processors and the total computational time is about half. If $\mathcal{N} >> Q$ many subproblems have to be solved on the same processor.

If, however, a subproblem does not fit into the memory, we additionally have to make use of the parallelization strategy from Section 3, partitioning the multi-dimensional CONV method for all the subproblems in a sparse grid layer. Let's consider, as an example, a 7D problem with $n_s = 10$ and $b = 2$. This problem requires 2^{28}-grid points for the subproblems in the top layer. The total number of subproblems in that layer is 1716, but these subproblems need to be partitioned once according to the splitting in Section 3. Therefore, we deal with 3432 subproblems of roughly 2^{27}-points. The full grid problem would require 2^{70}-grid points (2^{43} GB), which is infeasible. The overall sparse grid complexity is 2^{39}-points, subdivided into 5147 subproblems.

The accuracy of the CONV method can form the basis for some insight in the error for the sparse grid case. For a single-asset option with the asset modeled by geometric Brownian motion a second order full grid convergence was derived in [12]. We assume an error of $\mathbf{O}(\sum_{j=1}^{d} \Delta x_j^2)$ for the multi-dimensional problem. For the sparse grid integration technique, Gerstner [8] showed that the order of convergence is of $\mathbf{O}(\Delta x^2 \left(\log\left(\Delta x^{-1}\right)\right)^{d-1})$, for problems with bounded mixed derivatives. We assume here that this sparse grid error expansion is also valid for the max- and min-asset contracts evaluated in the next section. The accuracy for the sparse grid case depending on the maximum number of grid points in one direction ($N_S = 2^{n_s}$) can be related "globally" to the number of grid points $N_j = 2^{n_f}$ in the full grid case, as follows:

$$C_f 2^{-2n_f} = C_s 2^{-2n_s} n_s^{d-1}, \tag{43}$$

with constants C_f and C_s. The solution to this equation is given by:

$$n_s = \exp\left(-\mathcal{L}\left(-\frac{\ln 4}{d-1} e^{D}\right) - D\right)$$
$$D = \frac{-n_f \ln 4 + \ln C_f - \ln C_s}{d-1}$$

and $\mathcal{L}$ is the Lambert W function[1]. With this expression, we can compute the required number of grid points for a sparse grid computation to mimic a certain full grid problem with grid size n_f, given the desired accuracy ε, constant C_f and number n_f and the upper-bound of C_s. The constants C_f and C_s can be

[1]The Lambert W function is the solution of $\mathcal{L}(x)e^{\mathcal{L}(x)} = x$.

determined from a small-sized experiment taking into account that especially C_s is problem dependent.

4.1 Sparse grid computations

In the sparse grid method, we use the CONV algorithm as in the full grid case. In fact, the sparse grid method can be coded as an outer loop running over all subproblems. Within the loop, the CONV method is called with the desired grid parameters. We developed the algorithm so that if the subproblems in the sparse grid are additionally partitioned as described in Section 3, the number of parallel tasks increases to $U = \mathcal{N}B$, where B is the number of parts of a subproblem. The algorithm loops over all tasks U. Every task is sent to a different processor as soon as the processor is available. The maximum number of tasks in a problem is limited to 2^{31} on a 32-bit machine and 2^{63} on a 64-bit machine. As soon as the problem is solved for a certain $\mathbf{q}$ and multi-index $\mathcal{I}_{d,\ell}$, the solution (an option value) is returned to the master process and summed. After this task is performed, a new task can be assigned to this processor. This kind of parallel coding is not straightforward, due to the three different types of partitioning within the algorithm, but it can be used on a heterogeneous cluster.

We now perform numerical experiments with option contracts on the maximum or minimum of the underlying assets. The option parameters for these experiments are

- $K = 100$, $T = 1$ year, $r = 4.5\%$,.
- $\sigma_1 = 0.25, \sigma_2 = 0.35, , \sigma_3 = 0.20, , \sigma_4 = 0.25, , \sigma_5 = 0.20, , \sigma_6 = 0.21$ and $\sigma_7 = 0.27$.
- $\delta_1 = 0.05, \delta_2 = 0.07, \delta_3 = 0.04, \delta_4 = 0.06, \delta_5 = 0.04, \delta_6 = 0.03$ and $\delta_7 = 0.02$.
- $$R = \begin{pmatrix} 1.00 & -0.65 & 0.25 & 0.20 & 0.25 & -0.05 & 0.05 \\ -0.65 & 1.00 & 0.50 & 0.10 & 0.25 & 0.11 & -0.016 \\ 0.25 & 0.50 & 1.00 & 0.37 & 0.25 & 0.21 & 0.076 \\ 0.20 & 0.10 & 0.37 & 1.00 & 0.25 & 0.27 & 0.13 \\ 0.25 & 0.25 & 0.25 & 0.25 & 1.00 & 0.14 & -0.04 \\ -0.05 & 0.11 & 0.21 & 0.27 & 0.14 & 1.00 & 0.19 \\ 0.05 & -0.016 & 0.076 & 0.13 & -0.04 & 0.19 & 1.00 \end{pmatrix}.$$

with R the matrix with the correlation coefficients ρ_{jk}.

We start with the 4D problem and compare the sparse and the full grid results. The parameters for this experiment are listed above, where we take the first four subscript entries. In Table 4, the results for the full grid experiment are presented for a European and a Bermudan style contract. The Bermudan style contract has ten exercise dates during the lifetime of the option contract. The results are presented for grids with $N_j = 2^{n_f}$, $j = 1, \ldots, d$ points. We see a smooth convergence for this type of European contract and a accuracy better than €0.01 when $n_f = 7$. For the Bermudan contract, the convergence ratio

$d=4$	European			Bermudan		
n_f	price	error	Ratio	price	error	Ratio
3	0.38	7.38×10^{-1}		0.61	5.06×10^{-1}	
4	0.87	2.51×10^{-1}	2.94	0.53	9.25×10^{-1}	0.55
5	1.05	6.82×10^{-2}	3.67	1.87	3.36×10^{-1}	2.75
6	1.10	1.76×10^{-2}	3.88	1.85	2.88×10^{-2}	11.67
7	1.11	4.51×10^{-3}	3.90	1.84	5.37×10^{-3}	5.36

Table 4: European and Bermudan 4D put option on the maximum of the underlying assets on a full grid.The Bermudan contract has 10 exercise dates.

is less smooth, but the accuracy is again satisfactory. Although the results are satisfactory in the 4D case, the Bermudan option contract, for example, cannot be computed with $n_f = 7$ in higher dimensional cases without any form of communication. In Section 4, we derived an expression to compute the number of grid points $N_s = 2^{n_s}$ needed for mimicking the solution with sparse grid, given the accuracy and the number of grid points $N_f = 2^{n_f}$ in the full grid. The estimate of the number of grid points is very global and we assume the value of C_f as an upper bound for C_s. If the desired accuracy is $\approx 10^{-3}$, we need in the full grid $n_f = 8$ by applying the convergence ratio. From the results in Table 4, we have with $n_f = 7$ $C_f \approx 74$. Solving equation (43), the number of grid points for the sparse grid computation is $n_s = 13$, to have an accuracy of 10^{-3}. The choice of the base b in the sparse grid technique has a major influence on the complexity. The CONV method does not work if one of the coordinates is discretized in only two points, so $b \geqslant 2$. It is also reasonable for accuracy reasons to use a higher base [11], for example $b = 3$ which means at least 8 points per coordinate. This however has a significant impact on the costs of the method.

In Table 5, the results for the put option an the maximum of the underlying assets are presented in four and five dimensional case and for European and Bermudan style based on a sparse grid technique with $b = 2$. In this table, the first column represents the mimic of the full grid. With $n_s = 13$, we conclude that the desired accuracy of 10^{-3} is a proper choice. The sparse grid method also converges to the same value as the full grid case (see Table 4), although the convergence tends to be of first order and irregular. For the five asset problem (right part of Table 5), we see the same behavior of the four asset problem by means of accuracy and convergence. Finally, the Bermudan option contract also reaches the desired accuracy when $n_s = 13$.

The interesting point is the CPU time. In Table 4, we have an accuracy of 4.5×10^{-3} when $n_f = 7$ for the full grid European 4D option. The CPU time on 16 equivalent processors for the full grid problem (see Table 1) is 25 seconds. We have an accuracy of 4.68×10^{-3} with $n_s = 11$ in Table 6 for the same option, but now in sparse grid case. In Table 6, the parameters are presented for the 4D sparse grid case with $n_s = 11$. Each row in Table six represents a layer of the combination technique with the complexity, value of ℓ and number of subproblems. The right part gives the CPU times for each subproblem of

	European 4D			European 5D		
n_s	Price	Error	Ratio	Price	Error	Ratio
7	1.0488	5.25×10^{-2}		0.7407	3.48×10^{-2}	
8	1.0785	2.97×10^{-2}	1.77	0.7696	2.90×10^{-2}	1.20
9	1.0992	2.06×10^{-2}	1.44	0.7875	1.79×10^{-2}	1.62
10	1.1061	6.88×10^{-3}	3.00	0.7967	9.21×10^{-3}	1.94
11	1.1107	4.68×10^{-3}	1.47	0.8015	4.77×10^{-3}	1.93
12	1.1134	2.62×10^{-3}	1.79	0.8035	2.04×10^{-3}	2.34
13	1.1146	1.22×10^{-3}	2.15	0.8046	1.09×10^{-3}	1.34
	Bermudan 4D			Bermudan 5D		
n_s	Price	Error	Ratio	Price	Error	Ratio
10	1.830	1.34×10^{-2}	3.68	1.389	5.65×10^{-2}	0.32
11	1.838	5.09×10^{-3}	2.63	1.380	8.49×10^{-3}	6.66
12	1.840	3.18×10^{-3}	1.60	1.375	5.16×10^{-3}	1.65
13	1.841	2.56×10^{-3}	1.24	1.378	2.24×10^{-3}	2.30

Table 5: Sparse grid results of a 4D and 5D put option on the maximum of the assets

Problem parameters				CPU times		
j	ℓ	Complex.	#probl	Problem	Layer time	Q=12
1	17	2^{20}	165	0.51	82.5	7.3
2	16	2^{19}	120	0.24	28.8	2.5
3	15	2^{18}	84	0.12	10.1	1.0
4	14	2^{17}	56	0.05	2.88	0.3
Tot			425		124.2	11.1

Table 6: Problem parameters for sparse grid (mimic of the 4D 2^{11} full grid)

a specific layer, the layer's total sequential CPU time and parallel CPU time when 12 CPUs are used. The total time on a single computer is 124.2 seconds and on a heterogeneous cluster 11.1 seconds. The efficiency is 11.23, which is high when 12 is the ideal case. Also we see that the CPU time in the sparse grid on 12 CPUs is even lower than the CPU time for the full grid case on 16 CPUs (25 seconds, see Table 1). We conclude that the sparse grid technique is an efficient method to use in parallel on a low number of CPUs, whereas the problem size of the partitioned full grid is still large. For example, the problem size of a problem in the top layer of the 5D 2^{13} mimic in Table 5 has a total complexity of 2^{25} or 512 MB.

We conclude our sparse grid results with the higher dimensional examples of the option contracts on the maximum or minimum of the assets. In Table 7, the results of the sparse grid computation are presented for a put option on the maximum and minimum of the six or seven underlying assets. We again see a satisfactory accuracy with $n_s = 10$ and an irregular convergence. The 7D sparse grid problem with $n_s = 10$ uses the sparse grid technique as well as

the partition technique in Section 3, because the maximum available memory is 2GB on our heterogeneous cluster. The problem size of this experiment in the top layer is 4 GB and therefore it is partitioned with $B = 2$. Again the base of the sparse grid technique is set to $b = 2$. The hedge parameters can also be computed with the sparse grid technique. See the appendix for the results for Δ and Γ.

	6D Put on minimum			6D Put on maximum		
n_s	Price	Error	Ratio	Price	Error	Ratio
7	27.093	1.43×10^{-1}		0.375	2.33×10^{-2}	
8	27.183	9.02×10^{-2}	1.58	0.396	2.13×10^{-2}	1.09
9	27.141	4.21×10^{-2}	2.14	0.412	1.50×10^{-2}	1.42
10	27.158	1.73×10^{-2}	2.43	0.420	8.89×10^{-3}	1.69
	7D Put on minimum			7D Put on maximum		
n_s	Price	Error	Ratio	Price	Error	Ratio
7	26.153	1.22×10^{-1}		0.179	1.45×10^{-2}	
8	26.217	6.31×10^{-2}	1.93	0.194	1.50×10^{-2}	0.96
9	26.189	2.72×10^{-2}	2.32	0.206	1.14×10^{-2}	1.31
10	26.203	1.34×10^{-2}	2.02	0.213	7.21×10^{-3}	1.58

Table 7: Sparse grid results of two types of 6D and 7D European contracts

5 Conclusions

The multi-dimensional CONV method is a powerful and fast method. It is able to efficiently price multi-asset options of European and early-exercise type under Lévy price dynamics, including geometric Brownian motion, and to compute the hedge parameters. The partitioning of the method enables us to distribute some multi-dimensional splitted parts over a system of parallel computers, which speeds up the computation. Since we chose to avoid communication in the parallel system, and thus require some additional computation, the parallel efficiency is not optimal.

With the help of the sparse grid technique, we can climb in the number of dimensions of the multi-dimensional contracts. The size of the target problem depends, of course, on the number of parallel processors that are at one's disposal. An important remark is, however, that the basic sparse grid technique, without any enhancements, can only be successfully applied for certain payoffs, i.e., solutions should have bounded mixed derivatives. We show that the min- and max-asset options exhibit a satisfactory sparse grid convergence. The parallel efficiency of the sparse grid method is excellent, as each subproblem can be computed independently.

For *high*-dimensional problems it becomes necessary to combine the parallel sparse grid method with the parallel version of the CONV method.

A logical next step in our research is to evaluate the resulting parallel method on a machine containing a significant amount of parallel processors.

Acknowledgment: The authors would like to thank Roger Lord, Fang Fang and Hisham bin Zubair for valuable discussions and Kees Lemmens for his assistance with the parallel code.

References

[1] G. Bakshi and Z. Chen, An alternative valuation model for contingent claims. *J. Fin. Econometrics,* 44: 123-165, 1997.

[2] R. Bellman, *Adaptive Control Processes; a guided tour.* Princeton University Press, 1961.

[3] H.J. Bungartz, M. Griebel, D. Röschke, and C. Zenger, Pointwise convergence of the combination technique for the Laplace equation. *East-West J. Numer. Math.,* 2: 21–45, 1994.

[4] H.J. Bungartz and M. Griebel, Sparse Grids. *Acta Numerica,* 147–269, May 2004.

[5] P. Carr and D.B. Madan, Option valuation using the fast Fourier transform. *J. Comp. Finance* 2: 61-73, 1999.

[6] J. Gil-Pelaez, Note on the inverse theorem. *Biometrika* 37: 481-482, 1951.

[7] D. Gentle, Basket Weaving *Risk* 6:51-52, .

[8] T. Gerstner and M. Griebel, Numerical integration using sparse grids. *Num. Algorithms* 18: 209-232, 1998.

[9] R.M. Gray and J.W. Goodman, *Fourier Transforms.* Kluwer Dordrecht, Netherlands, 1995.

[10] J. Gurland, Inversion formulae for the distribution of ratios. *Ann. of math. statistics* 19: 228-237, 1948.

[11] C.C.W. Leentvaar and C.W. Oosterlee, *On coordinate transformation and grid stretching for sparse grid pricing of basket options* J. Comp. Appl. Math, to appear.

[12] R. Lord, F. Fang, F. Bervoets, and C.W. Oosterlee, *A fast and accurate FFT-based method for pricing early-exercise options under Lévy processes.* SSRN, page http://ssrn.com/abstract=966046, 2007.

[13] L. Scott, Pricing stock options in a jump-diffusion model with stocahstic volatility and interest rates: Application of Fourier inversion methods. *Math. Finance* 7: 413-426, 1997.

[14] S.J. Berridge and J.W. Schumacher, An irregular grid method for high-dimensional free-boundary problems in finance. *Future Gen. Comp. Systems* 20(3): 353-362, 2004.

[15] S.A. Smolyak, Quadrature and interpolation formulas for tensor products of certain classes of functions. *Dokl. Akad. Nauk. USSR* 4: 240-243, 1963.

[16] J. Wu, N.B. Mehta and J. Zhang, *A flexible approximation of the sum of log-normal distributed values* Glob. Telecom. Conf. (GLOBECOM), 6: 3413-3417, November 2005.

[17] C. Zenger. Sparse grids. In *Proceedings of the 6th GAMM seminar, Notes on numerical fluid mechanics*, volume 31, 1990.

[18] Y. Zhu, X. Wu, and I Chern. *Derivative securities and difference methods.* Springer Verlag, New York, 2004.

Appendix

Hedge parameters

The hedge parameters can be computed in an analytic way by the use of the derivative properties of the Fourier transform [9]:

$$\mathcal{F}\left(\frac{df}{dx}\right) = -i\omega\mathcal{F}\left(f\right).$$

Now the hedge parameters are:

$$\Delta_j(t,\mathbf{x}) = \frac{\partial V}{\partial S_j} = -\frac{e^{-r(T-t)}}{S_j}\mathcal{F}^{inv}\left\{i\omega_j\mathcal{F}\{V(T,\mathbf{y})\}\phi(-\boldsymbol{\omega})\right\}, \tag{44}$$

$$\Gamma_j(t,\mathbf{x}) = \frac{\partial^2 V}{\partial S_j^2} = \frac{e^{-r(T-t)}}{S_j^2}\mathcal{F}^{inv}\left\{\left(i\omega_j+\omega_j^2\right)\mathcal{F}\{V(T,\mathbf{y})\}\phi(-\boldsymbol{\omega})\right\}. \tag{45}$$

These equations can be discretized similarly to (27) into:

$$\Delta_k(t,\mathbf{x_m}) = -\frac{e^{-r(T-t)}}{(2\pi)^d S_k}\prod_{j=1}^{d}(-1)^{m_j}\mathcal{D}_d^{inv}\left[i\omega_{n_k}\phi_{\mathbf{n}}\mathcal{D}_d\left\{V_{\mathbf{k}}G_{\mathbf{k}}\right\}\right], \tag{46}$$

$$\Gamma_k(t,\mathbf{x_m}) = \frac{e^{-r(T-t)}}{(2\pi)^d S_k^2}\prod_{j=1}^{d}(-1)^{m_j}\mathcal{D}_d^{inv}\left[\left(i\omega_{n_k}+\omega_{n_k}^2\right)\phi_{\mathbf{n}}\mathcal{D}_d\left\{V_{\mathbf{k}}G_{\mathbf{k}}\right\}\right]. \tag{47}$$

	3D		4D		5D	
n_f	Δ_1	error	Δ_1	error	Δ_1	error
3	0.2077		0.1369		0.1087	
4	0.2118	4.13×10^{-3}	0.1370	8.90×10^{-4}	0.1119	3.29×10^{-3}
5	0.2120	1.30×10^{-4}	0.1371	1.16×10^{-4}	0.1116	3.40×10^{-4}
6	0.2119	9.56×10^{-5}	0.1372	2.22×10^{-5}	0.1116	1.90×10^{-5}

Table 8: Hedge parameters of a standard 3D,4D and 5D basket call on a full grid of 2^{n_f} points per coordinate (option parameters are in section 4.2.2)

	Full grid Put on the minimum			
n_f	Δ_1	error	Γ_1	error
3	-0.2821	1.27×10^{-1}	1.126×10^{-2}	2.91×10^{-3}
4	-0.2322	4.99×10^{-2}	1.024×10^{-2}	1.03×10^{-3}
5	-0.2232	8.99×10^{-3}	9.823×10^{-3}	4.12×10^{-4}
6	-0.2223	9.91×10^{-4}	9.765×10^{-3}	5.80×10^{-5}
7	-0.2222	9.97×10^{-5}	9.751×10^{-3}	1.47×10^{-5}
	Sparse grid Put on the minimum			
n_s	Δ_1	error	Γ_1	error
3	-0.2545		1.126×10^{-2}	
4	-0.2295	2.50×10^{-2}	9.262×10^{-3}	2.00×10^{-3}
5	-0.2209	8.63×10^{-3}	9.487×10^{-3}	2.25×10^{-4}
6	-0.2232	2.38×10^{-3}	9.661×10^{-3}	1.74×10^{-4}
7	-0.2211	2.18×10^{-3}	9.672×10^{-3}	1.14×10^{-4}
8	-0.2222	1.14×10^{-3}	9.774×10^{-3}	1.01×10^{-4}
9	-0.2224	2.14×10^{-4}	9.755×10^{-3}	1.86×10^{-5}
10	-0.2222	1.63×10^{-4}	9.752×10^{-3}	3.13×10^{-6}

Table 9: Hedge parameters for the 4D put option on the minimum of the assets in full and sparse grid