Efficient Parallel Implementation of a Weather Derivatives Pricing Algorithm based on the Fast Gauss Transform

Yusaku Yamamoto
Dept. of Computational Science & Engineering, Nagoya University
Furo-cho, Chikusa, Nagoya Aichi, 464-8603, Japan
yamamoto@na.cse.nagoya-u.ac.jp

Abstract

CDD weather derivatives are widely used to hedge weather risks and their fast and accurate pricing is an important problem in financial engineering. In this paper, we propose an efficient parallelization strategy of a pricing algorithm for the CDD derivatives. The algorithm uses the fast Gauss transform to compute the expected payoff of the derivative and has proved faster and more accurate than the conventional Monte Carlo method. However, speeding up the algorithm on a distributed-memory parallel computer is not straightforward because naïve parallelization will require a large amount of inter-processor communication. Our new parallelization strategy exploits the structure of the fast Gauss transform and thereby reduces the amount of inter-processor communication considerably. Numerical experiments show that our strategy achieves up to 50% performance improvement over the naïve one on an 16-node Mac G5 cluster and can compute the price of a representative CDD derivative in 7 seconds. This speed is adequate for almost any applications.

1. Introduction

Weather conditions such as air temperature, precipitation and snowfall have enormous impact on business activities. For example, higher-than-average temperature in summer will increase the revenue of air-conditioner makers and electric companies, but will reduce the profit of railway companies and department stores due to air-conditioning costs. As another example, less-than-normal snowfall in winter will save local governments the cost of removing snow, but may aversely affect the sales of hotels in skiing areas. Such variations in revenue or profit due to weather conditions are called weather risks.

To hedge these risks and stabilize the revenue, new financial instruments called weather derivatives have been developed and are widely traded in the market [4][6][9][12]. A weather derivative is a derivative security that gives the holder the right to receive a predetermined amount of money (called payoff) if certain weather conditions are met. For example, one can think of a temperature derivative that enables the holder to receive \$100 for each day in July for which the maximum temperature is below 20C. Such derivative can be used by air-conditioner makers to compensate for a possible loss of income due to a cold summer. Though weather derivatives are new financial products developed by Enron Corp. in 1997, its market size in Europe and the US has exceeded \$110 billion by 2002.

To sell a weather derivative, the issuer has to determine its rational price by constructing an appropriate stochastic model for future weather conditions and computing the expectation value of the payoff. Conventionally, the Monte Carlo method has been used to compute the expected payoff and several pricing systems based on this approach are now available [8]. However, as is well known, the convergence of the Monte Carlo method is quite slow. In fact, computing the price of a temperature derivative to 4-digit accuracy often requires more than 10^8 temperature scenarios and several minutes of CPU time on a modern PC. This is too long for real-time pricing, where it is desirable to get the result within a few seconds. In addition, there are applications such as the design of a customized weather derivative, in which one has to repeat the pricing over and over again with different input conditions to find the one that meets some specific needs of a customer. In that case, each pricing needs to be done in as short time as possible.

Recently, a new efficient pricing algorithm for temperature derivatives has been proposed [11][10]. In contrast to the Monte Carlo method, this method directly computes the probability distribution function (pdf) of the payoff using a recursion formula. Each step of recursion consists of multiple convolutions of functions with a Gaussian distribution and they are computed efficiently using the fast Gauss transform (FGT) [1][7], a variant of the fast multipole method proposed by Greengard and Strain. This algorithm has been applied to the CDD weather derivative, which is one of the most commonly used temperature derivatives, and has proved much faster and more accurate than the Monte Carlo method [11][10]. In addition, this algorithm has a large degree of concurrency because each convolution can be carried out in parallel.

In this paper, we present an efficient strategy for paralelizing this algorithm on a distributed-memory parallel computer. Though a preliminary parallel implementation is given in [10], it necessitates each processor to transfer about half of its data to other processors at each step of the recursion, as in the distributed-memory FFT algorithms. Thus the communication volume is very large and one cannot expect large speedup on parallel machines with slow inter-processor network, such as a PC cluster. In contrast, our new parallelization strategy exploits the structure of the fast Gauss transform and thereby reduces the amount of inter-processor data communication considerably. Furthermore, this makes it possible to overlap the communication with computation. The numerical experiments show that our new implementation achieves up to 50% performance improvement over the naïve one and attains 10.5 times speedup on a Power Mac G5 cluster with 16 nodes. We also believe that our parallelization strategy can be applied to other algorithms based on the fast multipole methods as well to enhance their parallel efficiency.

This paper is organized as follows: in Section 2, we formulate the pricing problem of CDD derivatives. Sections 3 and 4 present the FGT-based pricing algorithm and our new parallelization strategy, respectively. Numerical experiments that illustrate the effectiveness of our strategy are given in Section 5. Finally, Section 6 concludes the paper.

2 Mathematical formulation

2.1 Definition of a CDD derivative

In this paper, we deal with the CDD derivative, which is a temperature derivative most commonly traded in the market. In this section, we first give formal definitions of a generic temperature derivative and related technical terms and then introduce the CDD derivative.

To define a temperature derivative, one has to specify the following parameters:

- period of observation: N days in a specified period (e.g., from July 21st, 2006 to August 10th, 2006)
- point of observation: e.g., Tokyo
- weather index: W(C)
- strike value: S(C)
- tick: k (\$/C)
- contract type: *put* or *call*
- price of the derivative: Q (\$)

Here, the period of observation and the point of observation are the period and the point at which the temperature time series used to define the temperature derivative is observed. We denote the temperature on the n-th day during the period by T_n. The weather index W is a function of $\{T_1, T_2, \ldots, T_N\}$ and is used to define the payoff. Typical weather indices include the CDD, which we will define shortly, the average temperature $\frac{1}{N}\sum_{n=1}^{N} T_n$ and the maximum temperature $\max\{T_1, T_2, \ldots, T_N\}$. Using the weather index W, the strike value S and the tick k, the payoff P_{call} of a call option and P_{put} of a put option are defined, respectively, as follows:

$$P_{\text{call}} = k \cdot \max(W - S, 0), \quad (1)$$

$$P_{\text{put}} = k \cdot \max(S - W, 0). \quad (2)$$

A CDD derivative is a temperature derivative for which the underlying weather index is the CDD (Cooling Degree Days) defined by

$$CDD = \sum_{n=1}^{N} \max(0, T_n - \bar{T}). \quad (3)$$

Here, $\bar{T}$ is called the reference temperature. The payoff of the CDD derivative becomes larger as the number of days for which $T_n > \bar{T}$ is larger and the excess $T_n - \bar{T}$ is greater. Consequently, department stores and railway companies exposed to the risk of large air-conditioning costs due to higher-than-normal temperature in summer can hedge the risk by purchasing an appropriate CDD derivative.

2.2 The temperature model and principles for pricing

To find the rational price of the CDD derivative, one has to construct a stochastic model of the future temperature time series $\{T_1, T_2, \ldots, T_N\}$ and compute the expectation value of the payoff [9]. Several temperature models have been proposed for this purpose

[5][4][6]. Here we use the Dischel D1 model [6], which is widely used as a simple yet effective model. In this model, the daily temperature is assumed to evolve following the equation:

$$T_n = (1-\beta)\Theta_n + \beta T_{n-1} + \epsilon_n, \tag{4}$$

where β is a constant, Θ_n is the temperature of the n-th day in an average year (also a constant) and ϵ_n's are a sequence of i.i.d. random variables that follow the normal distribution $N(\mu, \sigma^2)$. The constants are determined from the observed data by least squares fitting and are sometimes adjusted to incorporate the long-range weather prediction.

Using the stochastic model (4), the price of the CDD call (or put) derivative is computed as

$$Q = E[P_{\text{call}}] + e, \tag{5}$$

where e is the premium determined by the issuer.

To compute the expectation value in (5), the conventional approach has been to use the Monte Carlo method. However, with the MC method, the error in the computed price decreases only as $\frac{1}{\sqrt{M}}$, where M is the number of sample paths. Thus it is often too slow for applications such as real-time pricing and design of a customized derivative, as we mentioned in the introduction, and a faster pricing algorithm is needed.

3 The FGT-based pricing algorithm

3.1 The basic idea

An alternative, more efficient algorithm for pricing CDD derivatives has been proposed by Yamamoto and Egi [11]. Though a complete description of the algorithm is given in [11] and [10], we present its summary here to use as a basis of discussion in the next section.

To compute the expectation value in (5), we first define the partial CDD on the n-th day by

$$C_n = \sum_{i=1}^{n} \max(0, T_i - \bar{T}). \tag{6}$$

By definition, the CDD is equal to C_N.

Let $P_n(T_n|T_{n-1})$ be the conditional probability density that the temperature on the n-th day is T_n under the condition that the temperature on the $(n-1)$-th day is T_{n-1}. Then, from eq. (4) we have

$$P_n(T_n|T_{n-1}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{(T_n-\mu_n)^2}{2\sigma^2}\right\}, \tag{7}$$

where

$$\mu_n = (1-\beta)\Theta_n + \beta T_{n-1} + \mu. \tag{8}$$

Next, let $p_n(T_n, C_n|T_{n-1}, C_{n-1})$ be the conditional probability density that the temperature and the partial CDD on the n-th day are T_n and C_n, respectively, under the condition that those on the $(n-1)$-th day are T_{n-1} and C_{n-1}, respectively. From the definition of C_n, we have $C_n = C_{n-1}$ if $T_n < \bar{T}$ and $C_n = C_{n-1} + (T_n - \bar{T})$ if $T_n \geq \bar{T}$. Hence $p_n(T_n, C_n|T_{n-1}, C_{n-1})$ can be computed as

$$\begin{aligned} &p_n(T_n, C_n|T_{n-1}, C_{n-1}) \\ &= \begin{cases} P_n(T_n|T_{n-1})\delta(C_n - C_{n-1}) & (T_n < \bar{T}), \\ P_n(T_n|T_{n-1}) \\ \quad \times\delta(C_n - (C_{n-1} + (T_n - \bar{T}))) & (T_n \geq \bar{T}). \end{cases} \end{aligned} \tag{9}$$

Here, $\delta(x)$ denotes Dirac's delta function.

From eqs. (7) and (9), we obtain the formulas for $p_n(T_n, C_n)$, the joint pdf of T_n and C_n, as follows [11]:

$$\begin{aligned} &p_n(T_n, C_n) \\ &= \begin{cases} \int_{-\infty}^{+\infty} dT_{n-1}\, P_n(T_n|T_{n-1}) \\ \quad \times p_{n-1}(T_{n-1}, C_n) & (T_n < \bar{T}), \\ \int_{-\infty}^{+\infty} dT_{n-1}\, P_n(T_n|T_{n-1}) \\ \quad \times p_{n-1}(T_{n-1}, C_n - (T_n - \bar{T})) & (T_n \geq \bar{T}). \end{cases} \end{aligned} \tag{10}$$

Equation (10) can be regarded as recursion formulas for $p_n(T_n, C_n)$. The initial conditions are

$$p_1(T, C) = \begin{cases} \delta(T - T_1)\,\delta(C) & (T_1 < \bar{T}), \\ \delta(T - T_1)\,\delta(C - (T_1 - \bar{T})) & (T_1 \geq \bar{T}). \end{cases} \tag{11}$$

Starting from eq. (11) and using eq. (10) repeatedly, we finally obtain $P_N(T_N, C_N)$, from which we can compute the expectation value of the payoff by

$$E[P_{\text{call}}] = \int_{-\infty}^{+\infty} dT_N \int_0^{+\infty} dC_N\; P_{call}\, P_N(T_N, C_N).$$

It would be appropriate to note that the initial pdf $p_1(T_1, C_1)$ has a δ-function like peak at $(T, C) = (T_1, 0)$ and this singularity is inherited by $p_n(T_n, C_n)$ $(n = 1, \ldots, N)$. To avoid numerical difficulties due to this, we divide $p_n(T_n, C_n)$ into the δ-function like part and the remaining regular part and integrate the former analytically. We omit the details here for brevity. Interested readers are referred to [11].

To compute eq. (10), we discretize T and C with step size h and approximate the integrals with some quadrature formula. Let $T^i \equiv \bar{T} + ih$ $(-M_T/2 \leq i \leq M_T/2)$, $C^j \equiv jh$ $(0 \leq j \leq M_C)$ and $p_n^{i,j} \equiv p_n(T^i, C^j)$, where

M_T and M_C are determined so that the joint pdf can be neglected outside the region $[-(M_T/2)h, (M_T/2)h] \times [0, M_C h]$. Then we can write the discretized version of the recursion formula (10) using an intermediate variable $\tilde{p}_n^{i,j}$ as follows:

$$p_n^{i,j} = \begin{cases} \tilde{p}_n^{i,j} & (i<0), \\ \tilde{p}_n^{i,j-i} & (i \geq 0), \end{cases} \tag{12}$$
$$(-M_T \leq i \leq j, \;\; 0 \leq j \leq M_C)$$

$$\tilde{p}_n^{i,j} = \sum_{k=-M_T/2}^{M_T/2} \frac{w^k}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{(T^i - \mu_n^k)^2}{2\sigma^2}\right\} p_{n-1}^{k,j},$$
$$(-M_T \leq i \leq M_T, \;\; 0 \leq j \leq M_C) \tag{13}$$

where w^k is the weight of the quadrature formula for the sample point T^k and

$$\mu_n^k = (1-\beta)\Theta_n + \beta T^k + \mu. \tag{14}$$

Based on eqs. (12) and (13), we can compute $p_n^{i,j}$ for $n = 1, 2, \ldots, N$ and find the expected value of the payoff from $p_N^{i,j}$.

3.2 Acceleration with the fast Gauss transform

The main task in the recursive computation of $p_n^{i,j}$ is the evaluation of the multiple sums in eq. (13). By examining this equation, we see that the computation for a fixed value of j has the form of discrete convolution of a sequence with a Gaussian distribution:

$$g_i = \sum_{k=1}^{N} q_k \exp\left\{-\frac{(x_i - y_k)^2}{\delta}\right\} \quad (1 \leq j \leq M). \tag{15}$$

Here, q_k, x_i, y_k and g_i correspond to w^k, $p_{n-1}^{k,j}$, T^i, μ_n^k and $\tilde{p}_n^{i,j}$, respectively. Direct computation of this convolution would require $O(M_T^2)$ arithmetic operations for each convolution, or $O(M_T^2 M_C)$ operations at each time step, which is a considerable work.

To reduce the computational work, we can apply the fast Gauss transform proposed by Greengard and Strain [7], which is a variant of the fast multipole transform designed to compute the convolution of eq. (15) efficiently. The key idea in the fast Gauss transform is to use the following truncated Taylor expansion with respect to both x_i and y_k:

$$e^{-\frac{(x_i-y_k)^2}{\delta}} \cong \sum_{\beta=0}^{\alpha_{\max}} \sum_{\alpha=0}^{\alpha_{\max}} \frac{1}{\beta!}\frac{1}{\alpha!}\left(\frac{y_k - y_0}{\sqrt{\delta}}\right)^{\alpha} \times h_{\alpha+\beta}\left(\frac{x_0 - y_0}{\sqrt{\delta}}\right)\left(\frac{x_i - x_0}{\sqrt{\delta}}\right)^{\beta}, \tag{16}$$

where $h_{\alpha+\beta}(x)$ is the Hermite function and $\alpha_{\max}$, x_0 and y_0 are constants. It can be shown that $\alpha_{\max} = 8$ is sufficient to achieve double-precision accuracy when $|(x_i - x_0)/\sqrt{\delta}| < 1/2$ and $|(y_k - y_0)/\sqrt{\delta}| < 1/2$ [7].

Now we consider a special case where all the target points $\{x_i\}$ are in an interval with center x_0 and length $\sqrt{\delta}$ and all the source points $\{y_k\}$ are in another interval with center y_0 and length $\sqrt{\delta}$ (See Fig. 1). Then, by substituting eq. (16) into eq. (15), we obtain the following expression for g_i:

$$g_i \cong \sum_{\beta=0}^{\alpha_{\max}} \frac{1}{\beta!}\left(\frac{x_i - x_0}{\sqrt{\delta}}\right)^{\beta} \left\{\sum_{\alpha=0}^{\alpha_{\max}} h_{\alpha+\beta}\left(\frac{x_0 - y_0}{\sqrt{\delta}}\right) \times \left\{\frac{1}{\alpha!}\sum_{k=1}^{N} q_k \left(\frac{y_k - y_0}{\sqrt{\delta}}\right)^{\alpha}\right\}\right\} \tag{17}$$

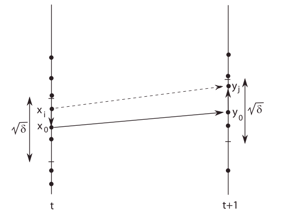


Figure 1. Illustration of the FGT algorithm.

This shows that the computation of g_i can be divided into three steps:

1. $A_\alpha \equiv \frac{1}{\alpha!}\sum_{k=1}^{N} q_k \left(\frac{y_k - y_0}{\sqrt{\delta}}\right)^{\alpha} \quad (0 \leq \alpha \leq \alpha_{\max})$.
2. $B_\beta \equiv \sum_{\alpha=0}^{\alpha_{\max}} A_\alpha h_{\alpha+\beta}\left(\frac{x_0 - y_0}{\sqrt{\delta}}\right) \quad (0 \leq \beta \leq \alpha_{\max})$.
3. $g_i = \sum_{\beta=0}^{\alpha_{\max}} B_\beta \frac{1}{\beta!}\left(\frac{x_i - x_0}{\sqrt{\delta}}\right)^{\beta} \quad (1 \leq i \leq M)$.

When $\alpha_{\max}$ is fixed, steps 1 and 3 require $O(N)$ and $O(M)$ computational effort, respectively, while step 2 can be done in a constant time that depends neither on N nor M. Thus the total computational work can be reduced to $O(M+N)$ from $O(MN)$.

In a general case, we divide the space of x-space and y-space into intervals of length $\sqrt{\delta}$ and apply the above algorithm to each of the interval-interval pair, regarding the centers of the intervals as x_0 and y_0. It can be shown that the total work is still $O(M+N)$ in this case [7] because each x_i and y_k belong to only one interval and the interaction between intervals that are far apart can be neglected due to the rapid decay of the Gaussian function.

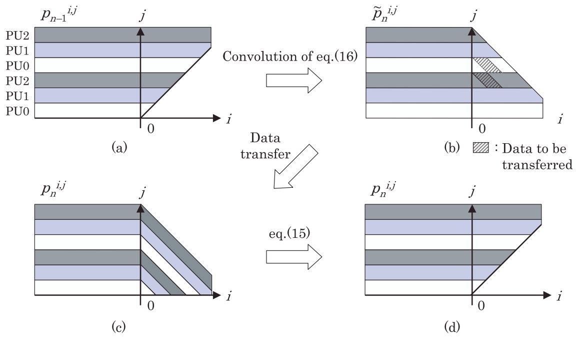
The use of the fast Gauss transform allows us to compute eq. (13) for each value of j in $O(M_T)$ work. Thus the total computational work for each time step is $O(M_T M_C)$. Furthermore, it can be shown that the pricing error of our algorithm decreases as $O(h^2)$, or as $O(1/(M_T M_C))$ [11]. Hence we can say that the error decreases inversely proportionally with the computational work and our algorithm has a convergence rate higher than that of the Monte Carlo method. We finally point out that the fast Gauss transform has proved useful in the pricing of other complex financial options such as the American options [2] and various path-dependent options [3].

4 An efficient parallelization strategy

Numerical experiments show that the pricing algorithm presented in the previous section is much more efficient than the Monte Carlo method and is up to ten times faster when computing the price of a CDD derivative to 4-digit accuracy [11][10]. Still, the pricing of a CDD derivative with a large number of monitoring dates takes more than 1 minute on a modern PC, so we need to seek an efficient strategy for parallelization.

As can be seen from eq. (13), the computation of $\tilde{p}_n^{i,j}$ for different values of j can be carried out independently. Thus the most natural way to parallelize the algorithm on a distributed-memory parallel machine would be to distribute the array $p_{n-1}^{i,j}$ among the processors in the j direction in a block cyclic fashion (Fig. 2 (a)) and have each processor compute the convolutions of eq. (13) for the values j allocated to it (Fig. 2 (b)). In fact, a preliminary parallel implementation based on this idea has been given in [10]. However, as is clear from eq. (12), $p_n^{i,j}$ for $i \geq 0$ is defined using $\tilde{p}_n^{i,j-i}$. Consequently, to have $p_n^{i,j}$ distributed in the same way as $p_{n-1}^{i,j}$, the element $\tilde{p}_n^{i,j}$ $(i \geq 0)$ computed by the processor taking charge of the j-th row has to be sent to another processor taking charge of the $(j+i)$-th row (Fig. 2 (c)). This causes all-to-all inter-processor communication in which every processor has to send about half of its data to other processors. When the number of processors is R, the number of elements to be sent by one processor at each time step is $M_T M_C/(2R)$. Since the computational work for each processor is also $O(M_T M_C/R)$, the ratio of communication volume to computational work is fairly high for this naïve parallelization strategy. Thus we cannot expect good speedup on parallel machines with slow inter-processor network, such as a PC cluster.

To reduce the volume of inter-processor communication, we consider exploiting the structure of the fast Gauss transform which we described in subsection 3.2. More specifically, we notice that the information needed to compute the outputs of FGT is aggregated in the coefficients $\{B_\beta\}$. The number of the coefficients $\{B_\beta\}$ for each interval is $\alpha_{\max}+1$ and is around ten, while the number of target points x_i in one interval is typically several dozens. Hence, if we can reorganize the parallel algorithm so that $\{B_\beta\}$ are communicated among the processors instead of $\tilde{p}_n^{i,j}$, the communication volume can be reduced considerably.

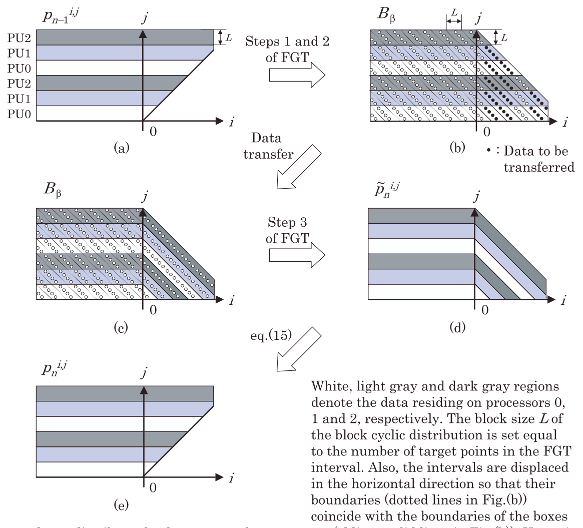


White, light gray and dark gray regions denote the data residing on processors 0, 1 and 2, respectively. Though $p_{n-1}{}^{i,j}$ is distributed in a block cyclic fashion, the distribution of $p_n{}^{i,j}$ becomes different as a result eq.(15). To have $p_n{}^{i,j}$ distributed in the same way as $p_{n-1}{}^{i,j}$. all-to-all inter-processor data transfer of $p_n{}^{i,j}$ is necessary prior to eq.(15).

Figure 2. Inter-processor data transfer in the old parallelization scheme.

To this end, we redefine the block size used in the block cyclic distribution and the intervals used in the FGT for each row. Let the block size be L (Fig. 3 (a)). We choose L equal to the number of target points in each interval and displace the intervals on each row so that their boundaries coincide with the boundaries of the blocks used to redistribute the data among the processors (Fig. 3 (b)). Then, each interval corresponds to the data to be sent to one processor and we can therefore send the intermediate coefficients $\{B_\beta\}$ associated with the interval (black circles in Fig. 3 (b)) instead of $\tilde{p}_n^{i,j}$. The processor that receives $\{B_\beta\}$ performs the step 3 of the FGT and obtains $\tilde{p}_n^{i,j}$ for that interval (Fig. 3 (c) and (d)).

More specifically, we denote the processor number by r $(0 \leq r < R)$, the set of rows allocated to processor r by J_r and the number of intervals used in the FGT for one row by N_{int}. From eq. (12), we see that the computation of $\tilde{p}_n^{i,j}$ for $i \leq 0$ requires no inter-processor communication. Considering this, we number the intervals on each row so that the leftmost interval, the leftmost interval that requires inter-processor communication and the rightmost interval are given numbers $-\bar{N}_{\text{int}}/2+1$, 1 and $\bar{N}_{\text{int}}/2$, respectively. We also define the l-th interval in the FGT for the j-th row as interval (j,l) and denote the coefficients $\{B_\beta\}$ associated with



White, light gray and dark gray regions denote the data residing on processors 0, 1 and 2, respectively. The block size L of the block cyclic distribution is set equal to the number of target points in the FGT interval. Also, the intervals are displaced in the horizontal direction so that their boundaries (dotted lines in Fig.(b)) coincide with the boundaries of the boxes used to redistribute the data among the processors (oblique solid lines in Fig.(b)). Hence it becomes possible to send the coefficients B_β(black circles in Fig.(b)) instead of $\tilde{p}_n^{i,j}$.

Figure 3. Inter-processor data transfer in the new parallelization scheme.

it by $\{B_\beta^{jl}\}$. With these notations, the operations of the r-th processor at one time step can be written as follows:

(i) Perform the steps 1 and 2 of the FGT for all $j \in J_r$.

(ii) Send $\{B_\beta^{jl}\}$ to processor mod$(r+l, R)$ for $l \geq 1$ and $j \in J_r$.

(iii) Receive $\{B_\beta^{j-L*l,l}\}$ from processor mod$(r-l, R)$ for $l \geq 1$ and $j \in J_r$.

(iv) Perform the step 3 of the FGT for the interval $(j - L * l, l)$ for $l \geq 1$ and $j \in J$.

(v) Perform the step 3 of the FGT for the interval (j, l) for $l \leq 0$ and $j \in J_r$.

In this parallelization scheme, the number of elements to be sent by one processor is $N_{\text{int}} M_C (\alpha_{\max} + 1)/(2R)$. Since the length of each interval is $\sqrt{\delta} = \sqrt{2}\sigma$ and is independent of the grid size h, N_{int} is a constant as long as the computational region is fixed. Also, $\alpha_{\max}$ is a constant. Thus we have succeeded in reducing the volume of inter-processor communication for each processor from $O(M_T M_C/R)$ to $O(M_C/R)$. Note that the new parallelization strategy has smaller communication volume if $N_{\text{int}}(\alpha_{\max}+1) < M_T$, or if the number of target points in each interval is greater than $\alpha_{\max}+1$.

It is interesting to point out that this new parallelization strategy creates an opportunity for overlapping communication with computation. To see this, we note that the step (v) above does not use the data received from other processor. We can therefore bring this step between steps (ii) and (iii), thereby allowing for more time before receiving the data.

The key observation in our new parallelization strategy has been that all the necessary information to get the final outputs of the FGT is aggregated in $\{B_\beta\}$ and it is therefore more efficient to communicate these intermediate coefficients instead of the final outputs. Since other fast transform algorithms based on the fast multipole method have a structure similar to that of the FGT, we believe that our idea can be applied to these algorithms as well.

5 Computational Results

5.1 Convergence of the FGT-based pricing algorithm

We implemented the algorithm described in sections 3 and 4 using C and MPI on a cluster of Mac G5. Each node has two 2.0GHz PowerPC G5 processor and 2Gbytes of memory, but we used only one processor per node because we wanted to evaluate the performance on a pure distributed-memory parallel machine. The nodes are connected via Gigabit Ethernet and up to 16 nodes were used in our experiments. We used GNU C++ compiler with '-fast' option and LAM MPI.

To evaluate the performance of our algorithm, we used CDD calls with the following parameters:

- period of observation: 10 or 20 days from July 7.
- point of observation: Tokyo
- weather index: CDD with reference temperature $\bar{T}$=24(C).
- strike value: S=20 and 40(C) for derivatives with N=10 and 20, respectively.
- tick: 1($/C)

The parameters for the Dischel model were $T_0 = 24$(C), $\beta = 0.7763$, $\mu = 0.0896$ $\sigma = 2.3734$, which were determined from observed data.

First, we compare the speed and accuracy of the FGT-based algorithm on a single processor with that of the Monte Carlo method. The results for the $N = 20$ case are shown in Fig. 4. Though a more detailed comparison (on the Alpha 21164A processor) using CDD derivatives with various values of N and S has been given in [10], here we present the results on the new PowerPC G5 processor for completeness. In the graph, the horizontal axis and the vertical axis denote the computational time and the computed price, respectively. The 95% confidence intervals are also shown for the Monte Carlo method. The value of M_T for the FGT-based method and the number of sample paths for the Monte Carlo method are also shown on the graph. M_C is set to $2M_T$ when $N = 10$ and to $4M_T$ when $N = 20$. As can be seen clearly from the graph, the convergence of the FGT-based method is much faster and smoother than the Monte Carlo method. In fact, the former computes the price to 4-digit accuracy in 27 seconds ($M_T = 960$), while the latter requires 165 seconds (using 10^8 sample paths). The results for other values of N and S were similar and are consistent with the results given in [10].

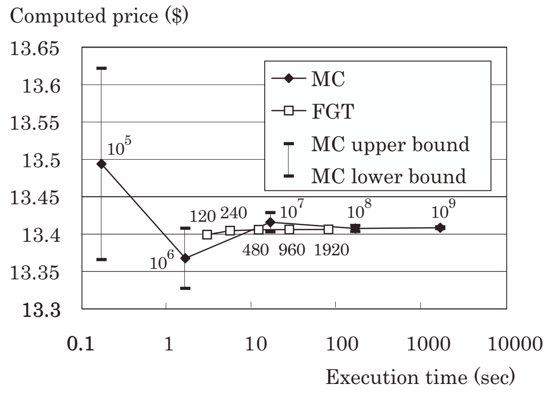


Figure 4. Convergence of the MC method and the FGT-based method for a CDD derivative ($N = 20$, $S = 40$).

5.2 Parallel performance

Next we study the performance of our new parallelization strategy proposed in section 4. We compare parallel implementations based on the following three strategies:

(a) The old parallelization strategy proposed in [10], which directly transfers the results of the FGT ($\tilde{p}_n^{i,j}$) among the processors.

(b) The new parallelization strategy proposed in Section 4, which transfers the intermediate results of the FGT (B_β) among the processors.

(c) Strategy (b) with overlapping of communication with computation.

The execution times on 1, 2, 4, 8 and 16 processors for various values of M_T are listed in Tables 1 and 2 for $N = 10$ and 20, respectively. The figures show that our new parallelization strategy can achieve considerable improvement over the old one, especially when the problem size M_T becomes larger and the number of processors increases. To confirm these observations, we show the parallel efficiency of each implementation on 8 and 16 processors as a function of the problem size in Fig. 5 for the $N = 20$ case. It is clear that the parallel efficiency of the old implementation decreases with the problem size. This is because the computational work and the communication volume is both $O(M_T M_C)$ and the efficiency of the FGT increases with M_T, leading to larger ratio of the communication time to the computation time. In contrast, the parallel efficiency of our new implementations increases with M_T. This is natural considering that its communication volume is $O(M_C/R)$ and therefore the ratio of communication to computation decreases with M_T.

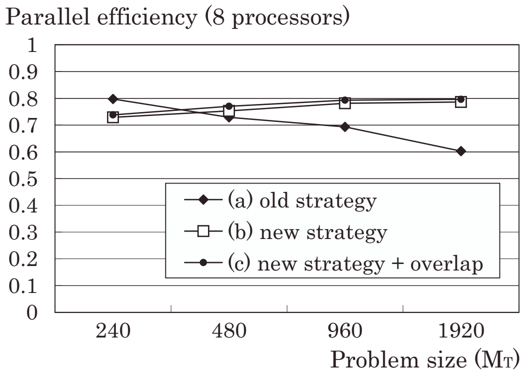
Table 1. Execution times (in sec.) of each implementation on the G5 cluster (N=10).

M_T	strategy	1	2	4	8	16
120	(a)	0.68	0.36	0.19	0.12	0.09
	(b)	0.68	0.38	0.21	0.13	0.10
	(c)	0.68	0.37	0.21	0.13	0.10
240	(a)	1.27	0.69	0.38	0.22	0.15
	(b)	1.27	0.70	0.39	0.23	0.17
	(c)	1.27	0.69	0.39	0.23	0.16
480	(a)	2.81	1.56	0.88	0.51	0.33
	(b)	2.81	1.53	0.87	0.49	0.33
	(c)	2.81	1.50	0.85	0.49	0.32
960	(a)	6.50	3.76	2.12	1.32	0.87
	(b)	6.45	3.47	1.90	1.12	0.72
	(c)	6.46	3.43	1.87	1.10	0.72

Table 2. Execution times (in sec.) of each implementation on the G5 cluster (N=20).

M_T	strategy	1	2	4	8	16
240	(a)	5.55	2.97	1.59	0.87	0.54
	(b)	5.54	3.00	1.66	0.95	0.60
	(c)	5.55	2.95	1.63	0.94	0.59
480	(a)	12.07	6.62	3.58	2.06	1.20
	(b)	12.05	6.47	3.53	2.00	1.20
	(c)	12.08	6.40	3.49	1.96	1.19
960	(a)	27.66	15.77	8.82	4.98	3.21
	(b)	27.45	14.58	7.86	4.39	2.64
	(c)	27.48	14.50	7.78	4.33	2.61
1920	(a)	79.94	48.68	28.93	16.58	10.97
	(b)	75.34	38.60	21.09	11.98	7.28
	(c)	75.61	38.28	20.81	11.87	7.20

From Table 1, we know that the effect of overlapping communication with computation is not large in this case. One of the reasons for this seems to be that in our new implementation (b), the inter-processor communication time is already small enough. We are planning to study the effect of overlapping in an environment with slower inter-processor network.



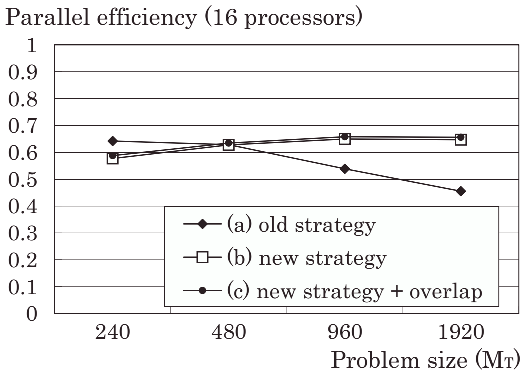


Figure 5. Parallel efficiency of the three implementations as a function of problem size M_T (8 & 16 processors, N=20).

Finally, we point out that when $M_T = 1920$, implementation (c) achieves 50% performance improvement over the implementation (a) and computes the price in about 7 seconds using 16 processors. This speed is adequate for almost all applications, including real-time pricing and design of a customized CDD derivative.

6 Conclusion

In this paper, we proposed an efficient parallel implementation of a pricing algorithm for CDD derivatives. The algorithm computes the probability distribution function of the CDD by repeating multiple convolutions of functions with a Gaussian distribution, and most of the computational time is spent in the fast Gauss transform. To parallelize this algorithm on a distributed-memory parallel computer, we devised a new strategy that reduces the inter-processor communication considerably by transferring the intermediate coefficients of the FGT instead of its outputs.

Numerical experiments show that our strategy achieves up to 50% performance improvement over the naïve one on an 16-node G5 cluster and can compute the price of a representative CDD derivative in about 7 seconds to 4-digit accuracy. Hence our implementation will be useful in applications such as real-time pricing and optimal design of a customized weather derivative, where the computation speed is critical.

Future work includes application to other types of weather derivatives and risk management of a portfolio consisting of a large number of weather derivatives.

Acknowledgements

I would like to thank Professor Shao-Liang Zhang and Professor Masaaki Sugihara for continuous support. I am also grateful to Takashi Uneyama and Takafumi Miyata for constructing and maintaining the Mac G5 cluster used in this work. This work is partially supported by Grant-in-Aid for the 21st Century COE Program "Frontiers of Computational Science".

References

[1] B. Baxter and G. Roussos. A new error estimate of the fast gauss transform. *SIAM Journal on Scientific Computing*, 24(1):257–259, 2002.
[2] M. Broadie and Y. Yamamoto. Application of the fast gauss transform to option pricing. *Management Science*, 49(8):1071–1088, 2003.
[3] M. Broadie and Y. Yamamoto. A double-exponential fast gauss transform algorithm for pricing discrete path-dependent options. *Operations Research*, 53(5):764–779, 2005.
[4] M. Cao and J. Wei. Pricing weather derivative: An equilibrium approach. Technical report, Department of Economics, Queen's University, Kingston, Ontario, Working Paper, 1999.
[5] J. S. D. Brody and M. Zervos. Dynamical pricing of weather derivatives. *Quantitative Finance*, 2:189–198, 2002.
[6] B. Dischel. The d1 stochastic temperature model for valuing weather futures and options. *Applied Derivatives Trading*, 1999.
[7] L. Greengard and J. Strain. The fast gauss transform. *SIAM Journal on Scientific and Statistical Computing*, 12(1):79–94, 1991.
[8] http://www.hitachi.co.jp/News/cnews/030424a.html.
[9] S. Jewson, L. Brix, and C. Ziehmann. *Weather Derivatives Valuation*. Cambridge University Press, 2005.
[10] Y. Yamamoto. An efficient and easily parallelizable algorithm for pricing weather derivatives. In *Proceedings of the LSSC'05*, to appear.
[11] Y. Yamamoto and M. Egi. Valuation of weather derivatives using the fast gauss transform (in japanese). *Journal of the Information Processing Society of Japan*, 45:176–185, 2004.
[12] L. Zeng. Pricing weather derivatives. *The Journal of Risk Finance*, 1(3):72–78, 2000.