High performance BLAS formulation of the multipole-to-local operator in the Fast Multipole Method

Olivier Coulaud, Pierre Fortin, Jean Roman

High performance BLAS formulation of the multipole-to-local operator in the Fast Multipole Method

(submitted)

Olivier Coulaud* Pierre Fortin* Jean Roman*

November 23, 2005

Abstract

The multipole-to-local (M2L) operator is the most time-consuming part of the far field computation in the Fast Multipole Method. Its natural expression, though commonly used, does not respect a sharp error bound: we here first prove the correctness of a second but less efficient writing. We then propose the use of BLAS (Basic Linear Algebra Subprograms) to speed up its computation for these two expressions. The more efficient level 3 BLAS are obtained through recopies but this additional cost can be avoided thanks to special data storages. This BLAS version is finally compared, theorically and practically with uniform distributions, to other M2L improvements such as block FFT and rotations. When considering runtime, extra memory storage and numerical stability, the BLAS version appears as the best one.

1 Introduction

The N-body problem in numerical simulations describes the computation of all pairwise interactions among N bodies. The Fast Multipole Method (FMM), developed by Greengard & Rokhlin [18] [17], solves this N-body problem for any given precision with $\mathcal{O}(N)$ runtime complexity against $\mathcal{O}(N^2)$ for the direct computation. The potential field is indeed decomposed in a near field part, directly computed, and a far field part approximated thanks to multipole and local expansions. First introduced for gravitational potentials in astrodynamics or electrostatic (coulombic) potentials in molecular dynamics [18] [12], it has then been extended with different mathematical bases to electromagnetism [7], VLSI capacitance [25], radiosity [28], object modeling [3] and many more. In this paper we will focus on gravitational and electrostatic potentials.

For these potentials, the $3D$ FMM has the drawback to present a linear complexity in the number of particles with an underlying factor in $\mathcal{O}(P^4)$ where P is the maximum degree in the expansions. In contrary the $2D$ FMM constant is in $\mathcal{O}(P^2)$. This limits the use of the FMM in $3D$ simulations and is particularly problematic for the multipole-to-local operator ($M2L$) which represents most of the runtime of the far field computation. Some schemes have been introduced in order to reduce this cost such as: Fast Fourier Transform (FFT) [13], rotations [32] and more recently plane wave expansions [5]. All these schemes reduce the $\mathcal{O}(P^4)$ factor to a $\mathcal{O}(P^3)$ or $\mathcal{O}(P^2)$.

We propose here a different approach: we will use highly efficient implementation techniques such as BLAS (Basic Linear Algebra Subprograms) [9] to improve the runtime of the FMM.

*ScAlApplix Project, INRIA Futurs and LaBRI UMR 5800; Université Bordeaux 1, 33405 Talence Cedex, France; olivier.coulaud@inria.fr, fortin@labri.fr, roman@labri.fr

Thanks to optimal use of the different layers of the hierarchical memory of the computer, so that the pipelines of the floating point units are filled at best, they indeed offer substantial runtime speedup on superscalar architectures. This speedup only affects the constant in the $\mathcal{O}(P^4)$ notation and we keep the $\mathcal{O}(P^4)$ operation count but since, in molecular dynamic simulations for example, P usually ranges from 3 to 15, which is quite low in terms of operation count, the speedup obtained with BLAS can exceed the one obtained with a scheme that offers a lower operation count.

We also distinguish two different expressions of the $M2L$ operator. Indeed, the error bound of the 3D FMM has been historically ([18] [17]) presented for the evaluation of potential with either finite multipole expansions or finite local expansions. But, as mentioned by several authors ([29], [31]), we have also to consider, when implementing the FMM, that the $M2L$ operator acts on finite multipole expansions, which means that both multipole and local expansions are finite. When denoting P the maximum degree of the expansions, and n (respectively j) the degrees of the multipole (respectively local) expansion terms, two different kinds of $M2L$ expressions can then be used. In the first one, both n and j fully range between 0 and P, whereas in the second $M2L$ expression we use only terms with: $n + j \leq P$. While the first one is natural and commonly used ([26], [29], [31]), no sharp error bound has yet been found, to our knowledge, for the corresponding summations. We will prove that the second one, though generally less efficient, respects such sharp error bound. For these two expressions, we will present the principles and the implementation features for two improvements of the $M2L$ operator, namely FFT and rotations, as well as for our BLAS approach. We will then propose a detailed comparaison of the memory requirements, numerical precisions and runtimes, for uniform distributions sequentially computed, between these three schemes depending on the $M2L$ expression used.

It has to be noted that the use of plane wave expansions [5] leads also to impressive speedup compared to the original formulation of the FMM. Since no free code of this new version is currently available, we are not able to compare it with the other schemes studied here. Nevertheless, it has to be noticed that this new version imposes tedious adaptation between the error introduced by the multipole and local expansions of the FMM and the error due to the use of plane waves: only a few precisions are thus currently available. The use of plane waves is yet promising and is still investigated (see [8]).

The rest of this article is organised as follows: in the next section we will introduce the formulae used in our implementation of the FMM. We will also discuss one error bound issue and present the two different versions of the $M2L$ operator. We will expose and discuss the FFT enhancement in section 2.3, the use of rotations in section 2.4, and the introduction of BLAS in section 3. All these improvements have been implemented in a code named FMB for Fast Multipoles with BLAS (or Fast Multipole in Bordeaux) thus allowing precise comparisons among them as presented in section 4. For almost all sections, more details and explanations can be found in [15]: the current article has to be seen as a summary of this research report.

2 Multipole to Local operator: presentation and current improvements

We present here the $M2L$ operator, its mathematical formulae, the two possible expressions, their consequence on the error bound and two current improvements of its operation count.

2.1 FMM presentation

We focus in this article on uniform distributions sequentially computed and refer the reader to the articles of Greengard & Rokhlin for a presentation of the FMM and especially the notions of: quadtree and octree, near and far fields, multipole and local expansions, upward and downward passes, nearest neighbors, *interaction list*, and *well-separateness*. We denote ws the well-separateness criterion (see [31]) and use $ws = 1$ here, which means that two cells of the octree are well-separated provided they do not share a boundary point. The interaction list of a cell c is

then defined as the set of all children of the nearest neighbors of the parent cell of c which are not themselves nearest neigbors of c: there is 189 members in each interaction list.

We denote by $P2M$ the operator that determines the multipole expansion of a leaf due to the particles it contains. $M2M$ denotes the translation of a multipole expansion, $M2L$ the conversion of a multipole expansion into a local expansion, and $L2L$ the translation of a local expansion.

We consider the root of the octree (or *computational box*) to be at level 0. The *height* of the octree is defined as its maximum level. Moreover we use Morton ordering for the indexing of each cell: this allows a fast access to all cells in the octree thanks to bit operations (see [12]).

We introduce now briefly the formulae used in our implementation of the FMM focusing on the $M2L$ operator: more details and the complete formulae set can be found in [15]. We have choosen the formulae of Epton & Dembart [14] for the clarity of the underlying theorems and their proofs, and also because they allow elementary formulation of the $M2L$ operator, which represents the most time-consuming part of the far field computation, while obeying the symmetry property among the multipole and local expansion terms of opposite order (see lemma 1 at the end of the section).

In $3D$ space, θ denotes the co-latitudinal coordinate and ϕ the longitudinal coordinate. With $\epsilon_m = (-1)^m$ if $m \geq 0$, and $\epsilon_m = 1$ otherwise, and P_l^m denoting the associated Legendre function, our unnormalized spherical harmonics Y_l^m of *degree* l and *order* m, with $l \geq 0$ et $-l \leq m \leq l$, are defined by:

$$Y_l^m(\theta, \phi) = \epsilon_m \sqrt{\frac{(l-m)!}{(l+m)!}} P_l^m(\cos\theta) e^{i.m.\phi}. \tag{1}$$

We emphasize that our spherical harmonics are considered as identically null for $l < 0$ or $|m| > l$.

Like Epton & Dembart [14], we respectively define the *Outer* and *Inner* functions by:

$$O_l^m(r, \theta, \phi) = \frac{(-1)^l i^{|m|}}{A_l^m} Y_l^m(\theta, \phi) \frac{1}{r^{l+1}} \qquad \forall\ (l, m) \in \mathbb{N}^2 \quad with \quad 0 \leq |m| \leq l, \tag{2}$$

and:

$$I_l^m(r, \theta, \phi) = i^{-|m|} A_l^m Y_l^m(\theta, \phi) r^l \qquad \forall\ (l, m) \in \mathbb{N}^2 \quad with \quad 0 \leq |m| \leq l, \tag{3}$$

where: $A_l^m = \frac{(-1)^l}{\sqrt{(l-m)!(l+m)!}}$.

Letting $\mathbf{X}$ and $\mathbf{X}'$ be two position vectors in $3D$ space, we now give the main theorems that the FMM requires: their proof can be found in [14].

Theorem 1 (Classical translation theorem) *Under the assumption $\|\mathbf{X}\| > \|\mathbf{X}'\|$, we have:*

$$\frac{1}{\|\mathbf{X} - \mathbf{X}'\|} = \sum_{n=0}^{+\infty} \sum_{l=-n}^{n} (-1)^n I_n^{-l}(\mathbf{X}') O_n^l(\mathbf{X}).$$

The next theorem is used to establish $M2M$ (*Outer-to-Outer*) and $M2L$ (*Outer-to-Inner*) operators.

Theorem 2 (Outer-to-Outer, Outer-to-Inner Laplace translation theorem) *Let $\|\mathbf{X}\| > \|\mathbf{X}'\|$, then:*

$$O_n^l(\mathbf{X} - \mathbf{X}') = \sum_{j=0}^{+\infty} \sum_{k=-j}^{j} (-1)^j I_j^{-k}(\mathbf{X}') O_{n+j}^{l+k}(\mathbf{X}) \qquad \forall\ (n, l) \in \mathbb{N}^2 \quad with \quad 0 \leq |l| \leq n.$$

The last theorem corresponds to the *Third Addition Theorem* in [17], and it is used for $L2L$ (*Inner-to-Inner*) operator.

Theorem 3 (Inner-to-Inner Laplace translation theorem)

$$I_n^l(\mathbf{X} - \mathbf{X}') = \sum_{j=0}^{n} \sum_{k=-j}^{j} (-1)^j I_j^k(\mathbf{X}') I_{n-j}^{l-k}(\mathbf{X}) \qquad \forall\ (n, l) \in \mathbb{N}^2 \quad with \quad 0 \leq |l| \leq n.$$

Thus, according to the classical translation theorem and the Outer-to-Inner Laplace translation theorem, we have the following expression for the $M2L$ operator that converts multipole expansion terms into local expansion terms, as defined in [15].

Proposition 1 (M2L operator) *Let M_n^l, with $n \geq 0$, $|l| \leq n$, the multipole expansion terms centered in $\mathbf{z_1}$, the local expansion terms L_j^k centered in $\mathbf{z_2}$, write :*

$$L_j^k = \sum_{n=0}^{+\infty} \sum_{l=-n}^{n} M_n^l O_{j+n}^{-k-l}(\rho, \alpha, \beta) \qquad \forall\ (j,k) \in \mathbb{N}^2 \quad with \quad 0 \leq |k| \leq j$$

where (ρ, α, β) are the spherical coordinates of the $M2L$ vector: $\mathbf{z_2} - \mathbf{z_1}$.

The $O_{j+n}^{-k-l}(\rho, \alpha, \beta)$ are named the *$M2L$ transfer functions* . The implementation of this formulae is named hereafter the *classic $M2L$*. Finally, we emphasize the following property which is also valid for the Outer and Inner functions, as well as for the multipole and local expansion terms: this enables the computation of only terms with positive orders in multipole and local expansions.

Lemma 1 (Symmetry among orders) *$Y_l^{-m} = \overline{Y}_l^m$ where $\overline{z}$ denotes the complex conjugate of $z \in \mathbb{C}$.*

2.2 Error bound analysis

As exposed in the introduction, the $M2L$ operator concretely uses finite multipole expansion to compute (finite) local expansions. When denoting P as the maximum degree of the expansions, whose terms have degrees therefore ranging from 0 to P, two different kinds of $M2L$ expressions can then be used.

The first one, named *M2L kernel*[1] *with double height*, corresponds to the outer sum on n, in the expression of the $M2L$ operator given in proposition 1, stopping at $n = P$, since in this case the O_j^k terms range up to $2P$. This one is natural and commonly used, but even if some interesting works have been done to estimate the behavior of the error induced by such $M2L$ expression (see [26] [29]), no sharp error bound has been found yet.

In the second $M2L$ expression the outer sum on n stops at $n = P - j$, so that the maximum degree of the O_j^k used is P. This is named *M2L kernel with single height*. This has been used for example in the DPMTA (Distributed Parallel Multipole Tree Algorithm) code developped at Duke University but the proof in the error bound given in [12] is wrong[2]. We however recommend the reading of this appendix since it presents a worst case error analysis similar to the one used hereafter. In particular it shows that no additional error is introduced by the $M2M$ operation. As explained in [17] the $L2L$ operation does not introduce any error at all. We can thus focus on the $M2L$ operation.

In the following we prove that an $M2L$ operator with single height kernel respects a sharp error bound (similar to the one presented in [12]). It has to be noted that this error bound for single height $M2L$ kernel was briefly presented by White & Head-Gordon [31], but they have used double height kernel in their implementation.

In order to prove this error bound, we study the potential in point $\mathbf{Z}$ due to one single unit charge located in $\mathbf{Q}$ as pictured in figure 1. We denote by $\mathcal{B}_1$ (respectively $\mathcal{B}_2$) the ball centered in $\mathbf{X_1}$ with radius r_1 (respectively X_2 and r_2). $\mathbf{Z}$ is enclosed in $\mathcal{B}_1$, $\mathbf{Q}$ in $\mathcal{B}_2$. Moreover, denoting $R = \|\mathbf{X_1} - \mathbf{X_2}\|$, we assume that: $R > r_1 + r_2$.

We first proceed as in [17]. Let α the angle between "vectors" $\mathbf{Q}$ and $\mathbf{Z}$, $r_>$ (respectively $r_<$) the maximum (respectively minimum) between the norm of $\mathbf{Q}$ and the one of $\mathbf{Z}$, and P_n the

[1] The *kernel* name has been inspired by [12].

[2] The equation (C.23) in appendix C of [12] is wrong because the composition of differential operators differs from the product of the derivatives.

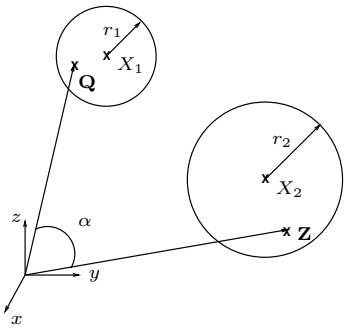


Figure 1: Our problem configuration.

Legendre polynomials, we have:

$$\Phi(\mathbf{Z}) = \frac{1}{\|\mathbf{Z}-\mathbf{Q}\|} = \sum_{n=0}^{+\infty} \frac{r_<^n}{r_>^{n+1}} P_n(\cos\alpha). \tag{4}$$

Defining $\Phi_P(\mathbf{Z})$ the potential with finite summation up to P, we have, since for all $x \in [-1,1]$, $|P_n(x)| \leq 1$:

$$\|\Phi(\mathbf{Z}) - \Phi_P(\mathbf{Z})\| \leq \frac{1}{r_> - r_<} \left(\frac{r_<}{r_>}\right)^{P+1}. \tag{5}$$

This leads to the following error bound:

Proposition 2 *$\forall\ \mathbf{Q} \in \mathcal{B}_1$, and $\forall\ \mathbf{Z} \in \mathcal{B}_2$, we have, under the condition $R > r_1 + r_2$:*

$$\|\Phi(\mathbf{Z}) - \Phi_P(\mathbf{Z})\| \leq \frac{1}{R - (r_1 + r_2)} \left(\frac{r_1 + r_2}{R}\right)^{P+1}.$$

Proof. Since $\|\mathbf{Z}-\mathbf{Q}\| = \|\left((\mathbf{Z}-\mathbf{X_2}) + (\mathbf{X_1}-\mathbf{Q})\right) + (\mathbf{X_2}-\mathbf{X_1})\|$ and since $\|\left(\mathbf{Z}-\mathbf{X_2}\right) + (\mathbf{X_1}-\mathbf{Q})\| \leq r_1 + r_2$, this results directly from the inequality (5).

We now decompose the potential in the same way the FMM does while maintaining this error bound, and prove the following theorem.

Theorem 4 *The M2L operator* **with single height** *M2L* **kernel** *strictly respects the error bound in proposition 2.*

Proof. We consider a multipole expansion, centered in $\mathbf{X_1}$, and a local expansion, centered in $\mathbf{X_2}$, to express the potential at $\mathbf{Z}$ due to the particle in $\mathbf{Q}$. Since the condition $R > r_1 + r_2$, which guarantees the convergence of both multipole and local expansions, implies $\|(\mathbf{Q}-\mathbf{X_1}) - (\mathbf{Z}-\mathbf{X_2})\| < \|\mathbf{X_2}-\mathbf{X_1}\|$, and since $I_n^l(-\mathbf{X}) = (-1)^n I_n^l(\mathbf{X})$ the potential in $\mathbf{Z}$ writes, with the classical translation theorem (theorem 1):

$$\Phi(\mathbf{Z}) = \sum_{n=0}^{+\infty} \sum_{l=-n}^{n} I_n^{-l}((\mathbf{Z}-\mathbf{X_2}) - (\mathbf{Q}-\mathbf{X_1}))O_n^l(\mathbf{X_2}-\mathbf{X_1}).$$

Thanks to the Inner-to-Inner translation theorem (theorem 3), we obtain:

$$\Phi(\mathbf{Z}) = \sum_{n=0}^{+\infty} \sum_{l=-n}^{n} \sum_{j=0}^{n} \sum_{k=-j}^{j} (-1)^j I_j^k(\mathbf{Q}-\mathbf{X_1}) I_{n-j}^{-l-k}(\mathbf{Z}-\mathbf{X_2}) O_n^l(\mathbf{X_2}-\mathbf{X_1}). \tag{6}$$

We emphasize here that the equation (6) is just a rewriting of equation (4): that is why when troncating the series in equation (6) for $n > P$, we obtain the same error bound as in proposition 2. Thus we have:

$$\Phi_P(\mathbf{Z}) = \sum_{n=0}^{P} \sum_{l=-n}^{n} \sum_{j=0}^{n} \sum_{k=-j}^{j} (-1)^j I_j^k(\mathbf{Q}-\mathbf{X_1}) I_{n-j}^{-l-k}(\mathbf{Z}-\mathbf{X_2}) O_n^l(\mathbf{X_2}-\mathbf{X_1}).$$

After an inversion of the two finite summations on the degrees n and j and reindexing $n-j \rightarrow n'$ and $l+k \rightarrow l'$ we obtain:

$$\Phi_P(\mathbf{Z}) = \sum_{j=0}^{P} \sum_{k=-j}^{j} \sum_{n'=0}^{P-j} \sum_{l'=-(n'+j)+k}^{n'+j+k} (-1)^j I_j^k(\mathbf{Q}-\mathbf{X_1}) I_{n'}^{-l'}(\mathbf{Z}-\mathbf{X_2}) O_{n'+j}^{l'-k}(\mathbf{X_2}-\mathbf{X_1}).$$

We remember here that $I_{n'}^{-l'}(\mathbf{x})$ imposes: $-n' \leq l' \leq n'$. In the same way, $I_j^k(\mathbf{x})$ imposes: $|k| \leq j$, that is to say: $j+k \geq 0$, and : $k-j \leq 0$. Moreover:

$$\begin{array}{lll} j+k \geq 0 & \Rightarrow & n'+j+k \geq n', \\ k-j \leq 0 & \Rightarrow & -(n'+j)+k = -n'+(k-j) \leq -n'. \end{array}$$

Under these conditions and after reindexing $-l' \rightarrow l$ and $n' \rightarrow n$ and another inversion of summations on the degrees, we obtain:

$$\Phi_P(\mathbf{Z}) = \sum_{n=0}^{P} \sum_{l=-n}^{n} \left(\sum_{j=0}^{P-n} \sum_{k=-j}^{j} (-1)^j I_j^k(\mathbf{Q}-\mathbf{X_1}) O_{n+j}^{-l-k}(\mathbf{X_2}-\mathbf{X_1}) \right) I_n^l(\mathbf{Z}-\mathbf{X_2}).$$

For all $\mathbf{Q}$ in $\mathcal{B}_1$, the multipole expansion terms $M_j^k, \forall\ (j,k) \in \mathbb{N}^2$ with $0 \leq |k| \leq j$, being defined by: $M_j^k = (-1)^j I_j^k(\mathbf{Q}-\mathbf{X_1})$, we have thus proved that:

$$\Phi_P(\mathbf{Z}) = \sum_{n=0}^{P} \sum_{l=-n}^{n} L_n^l I_n^l(\mathbf{Z}-\mathbf{X_2}),$$

where the L_n^l denote the local expansion terms, centered in $\mathbf{X_2}$, due to the unit charge located in $\mathbf{Q}$, and obtained thanks to the $M2L$ operator with single height $M2L$ kernel applied to the multipole expansion terms M_j^k as:

$$L_n^l = \sum_{j=0}^{P-n} \sum_{k=-j}^{j} M_j^k O_{n+j}^{-l-k}(\mathbf{X_2}-\mathbf{X_1}) \qquad \square$$

Since the error in proposition (2) is higher than the one due to the multipole expansion terms (see [19]), and since no additional error is introduced by the $M2M$ and $L2L$ operators, the error bound of the FMM with a single height kernel $M2L$ operator is therefore the same as in proposition (2).

It has to be noted that the condition of well-separateness, with either $ws = 1$ or $ws = 2$, is in fact more strict than $R > r_1 + r_2$ in order to ensure a fast enough convergence. With $ws = 1$ for example, we have, for a cell of side l: $r_1 = r_2 = \frac{\sqrt{3}}{2}$ and $R \geq 2$. Thus:

$$\|\Phi(\mathbf{Z}) - \Phi_P(\mathbf{Z})\| \leq \frac{1}{R-(r_1+r_2)} \left(\frac{\sqrt{3}}{2} \right)^{P+1} = \frac{1}{R-(r_1+r_2)} (0.866)^{P+1}.$$

As a result, this error bound can safely be used in the case of single height kernel in order to determine the value of P according to the required precision. Moreover since the double height kernel, compared to the single height kernel, only adds terms in the $M2L$ formula, it respects at

least this error bound but even adds precision with additional cost at runtime. The problem is that we cannot predict this gain in the accuracy of the FMM, and therefore we cannot compare single and double height kernels according to the tradeoff between theoretical accuracy and runtime. Only comparisons with practical accuracies will be possible as described in section 4.

We will now present, for both single and double kernel heights, two current improvements of the $M2L$ operator that reduce its theoretical operation count.

2.3 Fast Fourier Transform

The use of Fast Fourier Transform (FFT) in order to speed up the $M2L$ computation has been introduced by William D. Elliott ([12] or [13]). This work was performed only for single height $M2L$ kernel and the block version used to prevent numerical instability as written in DPMTA has restricted the possible values of P to multiples of the block size.

We have generalized this FFT improvement to both single and double height $M2L$ kernels and our block version can handle any value for P.

The FFT has been applied only to $M2L$ operator since this is the most time-consuming operator in the FMM, but it would be straightforward to derive FFT for $M2M$ (see [14]) and $L2L$. However, as explained in [12], the full computation with the three consecutive operators cannot be directly performed in Fourier space.

The theoretical formulae of the FFT scheme are rather complex, especially for the block version, and the implementation details are really specific and tedious: we refer the reader to [15] for these details and present here only the main ideas required by this scheme as well as its advantages and drawbacks.

The principle behind the use of FFT in FMM is to view $M2M$, $M2L$ and $L2L$ operators as $2D$ convolutions (or correlations) between two sequences: this is easily achieved with a rewriting of the expansion terms. This convolution will take place in *real* space. After having been processed by a forward $2D$ discrete Fourier transform (DFT), the two sequences are considered to be in *Fourier* space. The convolution in real space corresponds then to a point-wise product in Fourier space, and a backward discrete Fourier transform gives the same result in real space as the original convolution *provided that the two sequences are periodic*. As presented in [14] for $M2M$, we have to use additional null terms in order to build periodic sequences out of the multipole and local expansions and the $M2L$ transfer function: this process named *zero-padding* results in array of size $(2P+1)^2$ for each expansion in Fourier space.

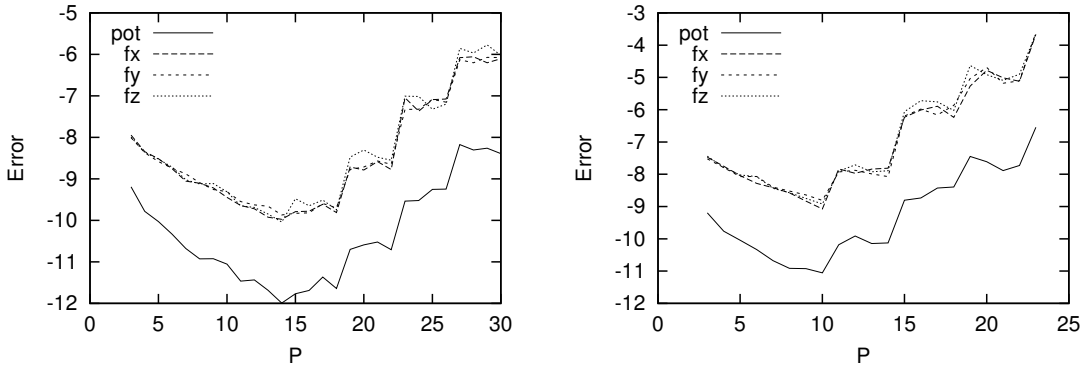
But the norm of the terms used in $M2L$ operation presents large varying magnitude for growing values of P. Due to limited machine precision this results with the FFT enhancement in numerical instabilities above a critical value for P that do not appear in the classic $M2L$ computation. This issue has partly been resolved in [13] with the use of a *block* FFT: in the degree dimension the terms are grouped according to their degrees, and hence according to the magnitude of their norm, and the $2D$ DFT is performed for every block seperately. The point-wise product has then to be performed block by block while using a *large grain convolution* (named as in [12]) among the blocks. The whole computation is however not really decomposable into blocks (as a matrix-product is, for example): the results are widespread among different blocks and a sum has thus to be performed on terms in these different blocks in order to retrieve each final local expansion term. When the number of degrees (i.e. $P+1$) is not a multiple of the size of the block, the first greater multiple is used here.

In order to have an efficient FFT implementation and one that is not restrained to powers of 2, we have used FFTW [16]. Nevertheless the symmetry property among orders (see lemma 1), which results in only half of the arrays actually stored in memory and used in the point-wise product, has forced us to split the $2D$ FFT in two sets of $1D$ FFT and perform recopies in between. Since no *wrap-around order* is used for the multipole expansion and the transfer function, the size of the halved arrays is finally $2(P+1)^2$ for each multipole expansion and each transfer function.

As for operation counts the most time-consuming part of the FFT scheme is the point-wise product which has to be computed 189 times (with $ws = 1$) per cell, against only one single forward FFT for the multipole expansion and one single backward FFT for the local expansion

of each cell. While each point-wise product runs in $\mathcal{O}(P^2)$ in the non-block version, the large-grain convolution of the block version leads to an operation count in $\mathcal{O}(P^3)$. This part of the computation does not match any of the standard BLAS calls (see section 3) and offers no data reuse: no special speedup can be expected from its implementation.

The block version does not however ensure complete numerical stability as shown in figure 2. For growing values of P this may be explained by the influence of the orders in the magnitude of the norm of the terms used: since the block version is applied only in the degree dimension, numerical instabilities arises in the order dimensions for high values of P. But numerical instabilities are also subject to the octree height used: this height indeed sets up the distance between two cell center in the $M2L$ transfer function terms, as well as the distance between a cell center and a particle location that appears in the multipole expansion terms. With increasing height of the octree, these distances decrease, since the cell size is divided by 2 between two consecutive levels of the octree, hence resulting in numerical instability when their order of magnitude is becomming too small. Of course this behaviour depends on the size of the computational box that encloses all particles, but with a bigger computational box the numerical instabilities would nevertheless appear for higher heights.



(a) Height 4

(b) Height 6

Figure 2: Logarithmic error for the potential (*pot*) and the force coordinates (fx, fy, fz) according to P for single height $M2L$ kernel with FFT with blocks of size 4. Tests performed on 10000 particles, uniformly distributed, with octrees of height 4 and 6: all particle coordinates are inside $[0, 1.0]$. The error plotted is the maximum absolute error over all the particles. Tests run on IBM Power3 and Power4 with double precision.

These instabilities can however be reduced with lower block size: while, for single height $M2L$ kernel, the FFT with block size 4 becomes unstable for P higher than 14 (see figure 2), a FFT with block size 3 is stable until 27, but is of course slower to compute because of the higher operation count due to the higher number of blocks.

The instability due to increasing octree height can also be avoided with a simple trick: when multiplying all particle cartesian coordinates, as well as the coordinates of the computational box, with a scalar $\alpha > 1$, the distances mentioned above are then increased which can reduce the numerical instability. The original potential and forces can be obtained afterward thanks to a multiplication with respectively $\frac{1}{\alpha}$ and $\frac{1}{\alpha^2}$. For example with $\alpha = 4$, the numerical instabilities with an height of 6 correspond to the ones formerly obtained with an height of 4. And with $\alpha = 4$ and an height of 4, no instabilities are detected in our tests for P between 3 and 40 with block size 4. This drawback however still exists and restrains the use of the FFT improvement for the FMM: the critical leaf size is a priori not known and depends on the machine precision used. Moreover whether this scaling has to be used or not depends on the value of P, the block size and the leaf size.

These instabilities appear for the same values of P and octree heights when running DPMTA (version 3.1.2p3). We have also compared the runtime of the FFT improvement in DPMTA (with fixed interaction list with $ws = 1$) and in our FMB: similar CPU times were measured which validates the efficiency of our implementation.

With double height $M2L$ kernel, the maximum degree of the O_j^k is $2P$: the sizes of the arrays storing the expansions in Fourier space are now $2(2P+1)^2$ for the $M2L$ transfer function, and the same has to apply to multipole and local expansions which is simply obtained with zero-padding. The FFT scheme for double height $M2L$ kernel operates then in the same way as for single height. But since O_j^k have now degrees up to $2P$, the range of magnitude is even greater and they become unstable for even lower values of P: this imposes even more the block version where we will have roughly twice more blocks than with single height $M2L$ kernel. As in the non-block version, the instabilities in the block FFT appear however for lower values of P than in the single height case. Finally as for operation counts, we focus in the block FFT on the large-grain convolution which is the most time-consuming part: the ratio between single and double height kernels is here 8, which clearly does not favor the double height kernel compared to the ratio of 6 in classic $M2L$ computation[3].

2.4 Rotations

The use of *rotations* has already been introduced in several articles: see [5], [6], [19], [20] and [32]. This improvement allows the computation of the $M2L$ operator[4] in $\mathcal{O}(P^3)$, against $\mathcal{O}(P^4)$ for their classic version. We have chosen to use the formulae detailed by Gumerov and Duraiswami in [20] since: they use the same definition for spherical harmonics, they use symmetries to speed up the computation and they focus only on the needed rotations. Moreover the recurrence is performed on real numbers, and not complex ones, and is simple to initiate. However no proof is given on the numerical stability of the formulae used.

The improvement in the use of rotations for $M2L$ operator is based on the fact that the cost of an $M2L$ operation performed along the $\mathbf{z}$ axis is $\mathcal{O}(P^3)$ against $\mathcal{O}(P^4)$ for general $M2L$. Indeed the associated Legendre functions verify: $P_n^m(1) = \delta_{m,0}$ where $\delta_{i,j}$ is the Kronecker delta symbol. We have therefore:

$$O_j^k(r,0,0) = \begin{cases} \frac{(-1)^j}{A_j^0 r^{j+1}} = \frac{j!}{r^{j+1}} & \text{if } k = 0, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

When the $M2L$ vector (defined in proposition 1) has spherical coordinates in the form $(r,0,0)$, the $M2L$ operator can be computed as:

$$L_j^k = \sum_{n=|k|}^{+\infty} M_n^{-k} O_{j+n}^0(r,0,0) \tag{8}$$

which results in $\mathcal{O}(P^3)$ operation count for both single and double height kernels.

For a general $M2L$ operation whose $M2L$ vector is not aligned with the $\mathbf{z}$ axis, we have first to rotate the Cartesian coordinates system so that so that the $M2L$ vector is along the $\mathbf{z}$ axis, then we perform the $M2L$ operation, and finally rotate back the coordinate system. The whole procedure is worthwhile since the first rotation on multipole expansion and the second on local expansion are both performed with $\mathcal{O}(P^3)$ operations. There indeed exists coefficients $T_j^{\nu,k}(\omega,\gamma,\chi)$, with $j \geq 0, |k| \leq j, |\nu| \leq j$, such that the spherical harmonics $Y_j^k(\theta,\phi)$ in the original coordinate system can be written with spherical harmonics $Y_j^\nu(\widehat{\theta},\widehat{\phi})$ in the rotated system as:

$$Y_j^k(\theta,\phi) = \sum_{\nu=-j}^{j} T_j^{\nu,k}(\omega,\gamma,\chi) Y_j^\nu(\widehat{\theta},\widehat{\phi}). \tag{9}$$

[3] For classic $M2L$ computation with single height $M2L$ kernel, we have, thanks to lemma 1, an operation count like: $\frac{1}{12}(P+3)(P+1)(P+2)^2$. With double height kernel, we have: $\frac{1}{2}(P+2)(P+1)^3$. Therefore the ratio among the two heights is roughly 6.

[4] This applies also to $M2M$ and $L2L$ operators.

These *rotation coefficients* can be extented to the Outer and Inner functions (see [15]), as well as to multipole and local expansions. They are computed according to a recursive computation performed in $\mathbb{R}$ which is described in [20] or [15]. Since, during the downward pass, the $M2L$ transfer functions are usually precomputed at each level for all $M2L$ vectors, these coefficients will be likewise precomputed: this step is therefore not critical at all in the runtime of the FMM and do not need to be accelerated.

The numerical stability of the recursive computation of rotation coefficients has been studied for example in [6] and [32] but for other recursive formulae than ours. We have thus check the accurary of the results in numerical tests on the whole FMM computation: no difference was detected between the classic $M2L$ and the $M2L$ with rotations, for values of P up to 30, for both single and double height kernels and for several octree heights.

When considering detailed operation count the ratio between single and double heights for $M2L$ kernel appears to be roughly $\frac{20}{7}$ which favors the double height kernel compared to the ratio of 6 of the classic $M2L$ implementation (see footnote 3). The memory requirements are very low: the only significant additional cost concern the results of the precomputation phase for each $M2L$ vector. Indeed, instead of the single $M2L$ transfer function for classic $M2L$ we now have to store the $M2L$ transfer function along the $\mathbf{z}$ axis, whose size grows like $\mathcal{O}(P)$, and two arrays containing the rotation coefficients for the two rotations: their sizes grow like $\mathcal{O}(P^3)$. As the number of $M2L$ vectors remains constant, the extra memory usage required by the rotation scheme remains small unless very low octree heights and very high values of P are used. In [15] we have also discussed the possibility of introducing BLAS (see section 3) in order to speed up the $M2L$ computation with rotations, but no efficient solution was found: while the rotations of multipole and local expansions cannot be written as matrix-vector products, the matrix-vector product corresponding to the $M2L$ operation along the $\mathbf{z}$ axis (whose operation count is lower than each rotation) computes too many useless terms and cannot be extended to matrix-matrix products.

In summary of section 2, we have shown that the single height $M2L$ kernel, contrary to the double height one, respects the FMM error bound which justify its implementation. Both block FFT and rotations schemes reduce the operation count, but the use of the block FFT, contrary to the use of rotations, requires important extra memory storage and may result in unpredictable numerical instabilities. We will now propose an alternative to speed up the $M2L$ computation.

3 Multipole to Local operator: implementation with BLAS

The BLAS (Basic Linear Algebra Subprograms) (see [9] [10] [23]) are a standard interface for some usual linear algebra operations such as a dot product of 2 vectors (level 1 BLAS), a matrix-vector product (level 2 BLAS) or a matrix-matrix product (level 3 BLAS). Optimized for the pipelines of the floating point units and for the hierarchical memory of the computer, the BLAS routines offer an efficient implementation of these operations, and the higher the level of the BLAS used, the higher the speedup they reach.

BLAS have already been used for hierarchical $\mathcal{O}(N)$ N-body algorithms by Hu and Johnsson [21] with Anderson's method [2] which uses different expansions than the FMM, and hence translation/conversion operators, but has the same algorithm for the upward and downward passes.

Here we propose the first BLAS implementation for the $M2L$ operator of the FMM for both single and double height kernel. In order to achieve the highest efficiency, we also detail several ways to use level 3 BLAS for $M2L$.

While the original FMM formulae of Greengard & Rokhlin (see [17]) do not allow a rewriting of their corresponding operators ($M2M$, $M2L$ or $L2L$) as matrix-vector products, this can be easily done with the simpler formulae of [12], [14] or [31]. The full FMM algoritm has also been rewritten in terms of matrix operations in [30]: here, we only focus on the rewriting of the $M2L$ operator as a matrix-vector product.

As we compute only terms with positive orders in the local expansion, we can write each $M2L$ operation as the following matrix-vector product, here written for $P = 2$ with single height $M2L$

kernel:

$$\left[\begin{array}{c} L_0^0 \\ \hline L_1^0 \\ L_1^1 \\ \hline L_2^0 \\ L_2^1 \\ L_2^2 \end{array}\right] = \left[\begin{array}{c|ccc|ccccc} O_0^0 & O_1^1 & O_1^0 & O_1^{-1} & O_2^2 & O_2^1 & O_2^0 & O_2^{-1} & O_2^{-2} \\ \hline O_1^0 & O_2^1 & O_2^0 & O_2^{-1} & 0 & 0 & 0 & 0 & 0 \\ O_1^{-1} & O_2^0 & O_2^{-1} & O_2^{-2} & 0 & 0 & 0 & 0 & 0 \\ \hline O_2^0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ O_2^{-1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ O_2^{-2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] . \left[\begin{array}{c} M_0^0 \\ \hline M_1^{-1} \\ M_1^0 \\ M_1^1 \\ \hline M_2^{-2} \\ M_2^{-1} \\ M_2^0 \\ M_2^1 \\ M_2^2 \end{array}\right] . \tag{10}$$

The computed vector is named *local vector*. The matrix is named $M2L$ *transfer matrix* and denoted $\mathbf{T}_{M2L}$. With double height $M2L$ kernel the matrix is dense: terms O_j^k with $j > P$ do not vanish as in the single height case. The building of this matrix, detailed in [15], is rather straightforward. The vector used is named *multipole vector*: terms with negative orders have to be stored.

3.1 Implementation with level 2 BLAS

Our matrices are stored in column storage mode, and we use the transfer matrix in its transposed form: it is indeed generally more efficient to compute $C \leftarrow A^T.B$ in row storage mode for A, than $C \leftarrow A.B$ with column storage mode for A; indeed, when considering the BLAS implementation (see [1] for a default implementation), the first solution leads to less writings, with however more readings, than the second one.

3.1.1 Double height $M2L$ kernel

The matrix $\mathbf{T}_{M2L}$ is dense: the use of level 2 BLAS $ZGEMV$ routine is therefore obvious, and this routine will automatically optimize the computation of the matrix-vector product to the underlying superscalar architecture. In order to differentiate it from the other BLAS method that will be used later, we name it *full_blas*.

3.1.2 Single height $M2L$ kernel

The matrix $\mathbf{T}_{M2L}$ is now sparse: we therefore have to split the sparse matrix-vector product in several dense block products.

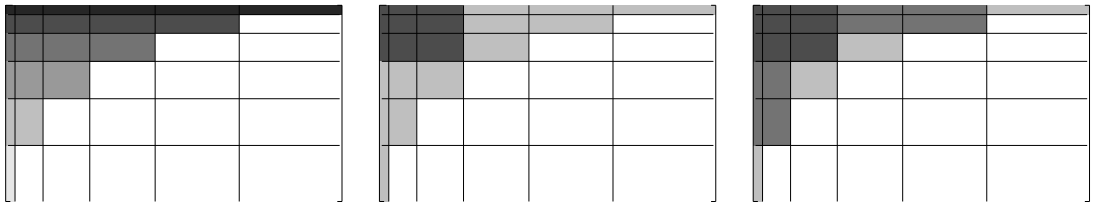
We refer the reader to the BLAS literature (see [11] for example) for the underlying techniques used to fill at best the pipelines of the floating point execution units in order to reach peak performance of the processor. These techniques are mainly: loop ordering for best spatial and temporal locality among data, loop blocking in order to provide maximum cache reuse, temporary copies in local arrays for problems with "leading dimensions" of the matrices (see BLAS routine interfaces), loop unrolling, register blocking, data prefetch, etc.

In [22], it has been shown that level 3 BLAS routines can efficiently be implemented only with the $GEMM$ level 3 BLAS routine and a few level 2 BLAS routines. For portability purpose, as well as for simplicity, we prefer to adopt such approach. We will thus decompose the sparse matrix-vector product corresponding to $M2L$ operator in several $ZGEMV$ block products in the most efficient way. $ZGEMV$ routine is used here since we treat a matrix-vector product, but in section (3.2) we will see how to use matrix-matrix products, and $ZGEMM$ routine will then be used as in [22]. All the optimizations recalled above will be used but left as much as possible to the underlying (and machine dependent) $ZGEMV/ZGEMM$ BLAS routine called. We point out now one important fact: in our implementation of the FMM, we are free in the memory storage of the blocks of $\mathbf{T}_{M2L}$ since its construction is precomputed at each level of the octree in the downward pass of the FMM. Therefore, most of problems that arise when considering the leading dimension of a matrix will be irrelevant with our blocks. With these principles in mind, we will

present an efficient block decomposition, named *block_blas*, for the matrix $\mathbf{T}_{M2L}$ with single height $M2L$ kernel.

In [22], triangular matrix-matrix product was performed with $ZGEMM$ routines thanks to a decomposition of the triangular matrix in *strips*. First, we proceed similarly using strips in the horizontal direction since the transfer matrix is stored by rows, and since this leads to several traversals of the multipole vector (one for each strip) against one single traversal on the local vector. In other words, with strips in the row direction, the resulting blocks of the local vector are updated one after the other, whereas the corresponding data of the multipole vector may be reloaded several times during the block matrix-vector product. As the local vector is traversed for updating (i.e. both reading and writing) and the multipole expansion for reading, this is more efficient than vertical strips which lead to the contrary. In practice, our strips strictly[5] respect the underlying structure of $\mathbf{T}_{M2L}$ as pictured on figure 3(a): each strip is separately stored and separately computed with one dedicated call to $ZGEMV$ routine. One obvious problem is that, as P grows, the first strips are too thin and too long while the last ones are too thick and too short: this can prevent the BLAS routine to internally decompose those strips according to the hierarchical memory of the computer.

But the underlying structure of $\mathbf{T}_{M2L}$ can be considered as recursive for a given P. Indeed when isolating the "biggest upper left square" (in the number of underlying sub-blocks) sub-matrix, with half of the sub-blocks in both the row and the column dimensions, as pictured in figures 3(b) and 3(c), we are left with one sub-matrix "on the right" and another one "below" whose sub-blocks are disposed in the same way as $\mathbf{T}_{M2L}$ for $\lfloor \frac{P}{2} \rfloor$. Detailed expression of this recursion can be found in [15].



(a) (b) (c)

Figure 3: *block_blas* decomposition for $P = 5$. (a) Decomposition with only strips: the different gray values show the different strips. (b) The "biggest upper left square" sub-matrix of $\mathbf{T}_{M2L}$ for $P = 5$ is pictured with the darker gray value. (c) In the remaining sub-matrices "on the right" and "below", whose sub-blocks are disposed in the same way as $\mathbf{T}_{M2L}$ for $P = 2$ (see equation (10)), we can recursively isolate a "biggest upper left square" sub-matrix.

Hence we use as much as possible dense matrices (i.e. the upper left sub-matrices) which are efficiently treated by $ZGEMV$ routine. With small values for P, which corresponds also to the terminal cases of the recursion, it may be not worth isolating this dense matrix: in these cases we use our strips. The inefficiency due to the shape of the first and the last strips is therefore minimized since we reserve the strips for the terminal cases. The terminal value for the recursion, hereafter named s_{block}, will have to be determined experimentally.

3.2 Level 3 BLAS

Since matrix-matrix product requires $\mathcal{O}(N^2)$ memory storage relatively to $\mathcal{O}(N^3)$ operation count, it is easier to overlap memory latency with computation of the floating point execution units with

[5] Another decomposition that uses additional zeros to obtain strips with optimal number of rows is discussed in [15].

level 3 BLAS than with level 2 BLAS, and thus to reach peak performance of the processor. We will therefore try to group multiple $M2L$ operations in one single matrix-matrix product.

During the downward pass, at a given level of the octree, all $M2L$ operations that have the same $M2L$ vector (see proposition 1) share the same $M2L$ transfer matrix. When considering all pairs of multipole and local expansions that share the same $M2L$ vector, it is thus possible to concatenate all their multipole vectors as columns of one single *multipole matrix* $\mathbf{M}^M$. The local vectors are also concatenated according to the same order to form one single *local matrix* $\mathbf{M}^L$. Then the matrix-matrix product $\mathbf{M}^L = \mathbf{T}_{M2L} \,.\, \mathbf{M}^M$ computes the corresponding $M2L$ operations all at once as described in figure 4. The concatenation is easily achieved thanks to the column storage of our matrices.

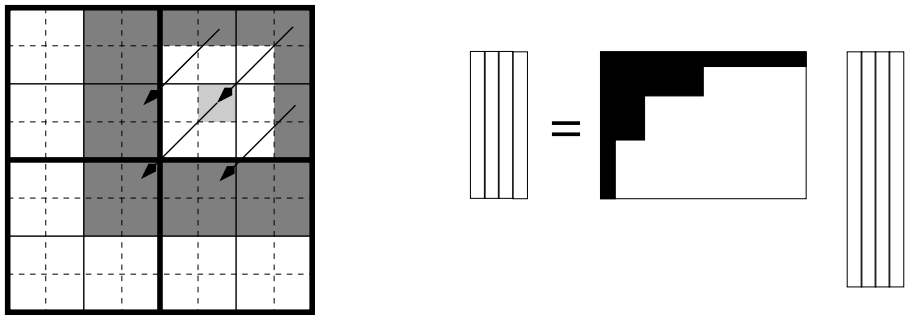


Figure 4: Concatenation of multipole and local expansions in order to use level 3 BLAS. The interaction list of the cell in light gray is marked with dark gray. Each of its $M2L$ operations is computed here with 3 other $M2L$ operations, that share the same $M2L$ vector, in one single matrix-matrix product: the corresponding multipole and local vectors are therefore concatenated in one multipole matrix and in one local matrix, both with 4 columns. The transfer matrix represented here is for a single height $M2L$ kernel.

First, we will roughly consider that the best efficiency is obtained with level 3 BLAS when the maximum number of multipole and local expansions are concatenated each time (see section 3.2.3 for revisions of this assertion). In the following, we will thus see how to concatenate the maximum number of multipole and local expansions for one given $M2L$ transfer matrix, and we will then present the implementation of this matrix-matrix product thanks to level 3 BLAS calls.

With free-space boundary conditions[6] (FBC), where all the space outside of the computational box is considered as empty, the cells located at the boundaries of the computational box, in each level, have an incomplete interaction list with less than 189 members for a *well-separateness* criterion $ws = 1$ (see section 2.1). In order to ease their computation, and for efficiency purpose, these cells with incomplete interaction list are computed separately when using level 3 BLAS for $M2L$.

3.2.1 Computing the local expansions of cells with incomplete interaction list

With free-space boundary conditions, the "cells with incomplete interaction list" are all the cells whose parent have at least one neighbor that is outside of the octree. In order to treat all the $M2L$ operations for the local expansions of such cells, we use copies of the multipole and local vectors. The $M2L$ computation is thus performed as: $\mathbf{M}^L \leftarrow \mathbf{T}_{M2L} \,.\, \mathbf{M}^M + \mathbf{M}^L$ and the columns of the local matrix are then *recopied* in the original local expansions.

For a given ws value, the $M2L$ vectors of the interaction list of a given cell are determined according to the *type of child* of the cell which describes the location of the cell center relatively to the center of its father. In $3D$, there is eight different possible *type of child*. Some $M2L$ vectors, but not all of them, are shared by all the *types of child*. We first loop on each possible $M2L$

[6]Such specificity does not appear with "Periodic Boundary Conditions", where the computational box is periodically replicated in each dimension.

vector and secondly, inside this first loop, we loop on each *type of child* that matches the current $M2L$ vector. This has been prefered to the opposite ordering of these two loops since it leads to a greater number of vectors treated per matrix-matrix product.

3.2.2 Computing the local expansions of cells with complete interaction list

All cells with complete interaction list have the same interaction list size, and all cells of the same *type of child* share exactly the same $M2L$ vectors. This regularity allows well suited algorithms.

The most simple uses each time *recopies* as in [21] in order to treat altogether the maximum number of pairs of multipole and local expansions. Contrary to the cells with incomplete interaction list, we first loop on each *type of child* and secondly loop on each $M2L$ vector corresponding to the current *type of child*. $\mathcal{V}_{M2L}$ denoting the set of all possible $M2L$ vectors, $\mathcal{T}_C$ the set of all different *types of child* and $\mathbf{T}_{M2L}(v)$ the $M2L$ transfer matrix corresponding to the $M2L$ vector v, this can be written as:

```
- ∀ t ∈ T_C do:
   - copy all local vectors in M^L ;
   - ∀ v ∈ V_M2L corresponding to t do:
      - copy the corresponding multipole vectors in M^M ;
      - perform M^L = T_M2L . M^M ;
     end do
   - copy back all the local vectors ;
  end do.
```

Thanks to the regularity, the *recopies* of the local expansions can be here performed out of the second loop, and the opposite order of the loops would lead to more copies for the local vectors while having the same number of vectors treated each time.

It is however possible to avoid the additional cost of *recopies* thanks to special *data storages* of our multipole and local vectors which are obtained through a "rearrangement" of these vectors in memory performed before the downward pass. In the first data storage, named *row* storage and illustrated in figure 5, we store consecutively in memory all the local vectors that belong to cells of the same *type of child* along a row in one given dimension of the $3D$ space. This is done for each row of the level, and the same storage is also done for the multipole vectors. Note however that for local vectors, only cells with complete interaction list have their local expansions rearranged in rows, while for the multipole expansions the rearrangement has to be performed for all cells of the level. With such data storage, we can call a level 3 BLAS routine starting at the local vector of the first cell of each row, with the corresponding $M2L$ transfer matrix and the corresponding multipole vector. It can be noted that Morton ordering (see section 2.1) is used here in order to access cells according to the coordinates of their center. With FBC, at level l and for a given *type of child*, the local expansions are hence rearranged in $(2^{l-1} - 2.ws)^2$ rows of size $2^{l-1} - 2.ws$.

The size of the rows corresponds to the number of local vectors treated per level 3 BLAS call; this might be not enough to obtain the best efficiency: for example there is only 6 columns at level 4. That is why we propose a second data storage, named *slice* storage, that stores consecutively the rows in memory in order to form a *slice* and thus enables several rows to be treated with one single level 3 BLAS call. In order to have a correct correspondence between slices of local expansions and slices of multipole expansions, we have to insert *blank boxes* between rows of local expansions: these corresponds to cells not belonging to the octree whose local vectors are computed uselessly. With $ws = 1$, there is 1 *blank box* at the beginning and 1 at the end of each row, as pictured in figure 6. We can however skip the first *blank box* of the slice (that is to say, the first *blank box* of the first row), and also the last one. With FBC, at level l, and for each *type of child*, the local expansions are rearranged in $2^{l-1} - 2.ws$ slices of size $(2^{l-1} - 2.ws)^2 + 2 * (2^{l-1} - 2.ws) - 2$. If *slice* storage allows a better efficiency with the level 3 BLAS, its clear drawback is that it also performs useless extra work due to the *blank boxes*.

This can even be extended to a third data storage, named *level* storage, resulting in concatenation of slices so that the whole level can be computed in one single BLAS call: in this case rows of *blank boxes* have to be added between each slice.

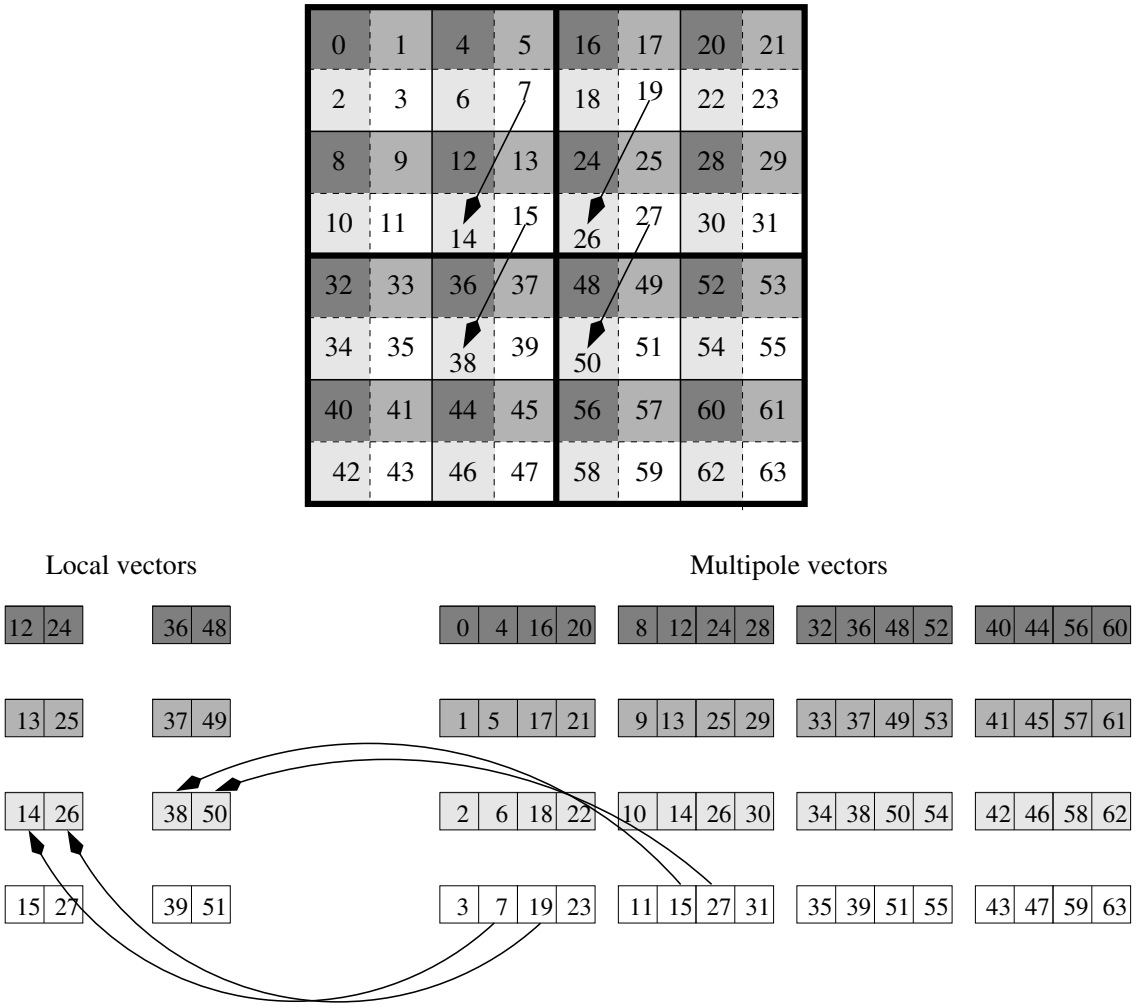


Figure 5: *row* storage ($2D$, $ws = 1$, level 3). The cells are indexed according to Morton ordering. Each of the 4 *types of child* has a different gray value. The expansions of the cells with same *type of child* along a given row are concatenated: the four $M2L$ operations, whose $M2L$ vectors are represented on the quadtree, can then be directly computed with matrix-matrix products (level 3 BLAS) without *recopies*.

3.2.3 Implementation with level 3 BLAS calls

As in section 3.1.1, the dense matrix-matrix product for double height $M2L$ kernel can be directly implemented with one single level 3 BLAS call: the $ZGEMM$ routine.

For single height $M2L$ kernel the decomposition proposed in section 3.1.2 is also valid with matrix-matrix products when replacing $ZGEMV$ routine by $ZGEMM$ one. However, we have to face a new constraint since we have several level 3 BLAS calls per matrix-matrix product: if all the columns of the multipole or local matrix cannot be stored in a level of the hierarchical memory (cache L1, L2 or even L3), or need more memory pages than the TLB (Translation Lookaside Buffer) can address, the whole matrix would have to be reloaded at each BLAS call. A matrix with $NbExp$ expansions is therefore split in sub-matrices with a constant number of columns, namely $NbExp_{max}$, and the same number of rows as the original matrix as described in figure 7: each sub-matrix is then treated separately. This decomposition of the matrix-matrix product is hereafter named "$NbExp_{max}$ decomposition".

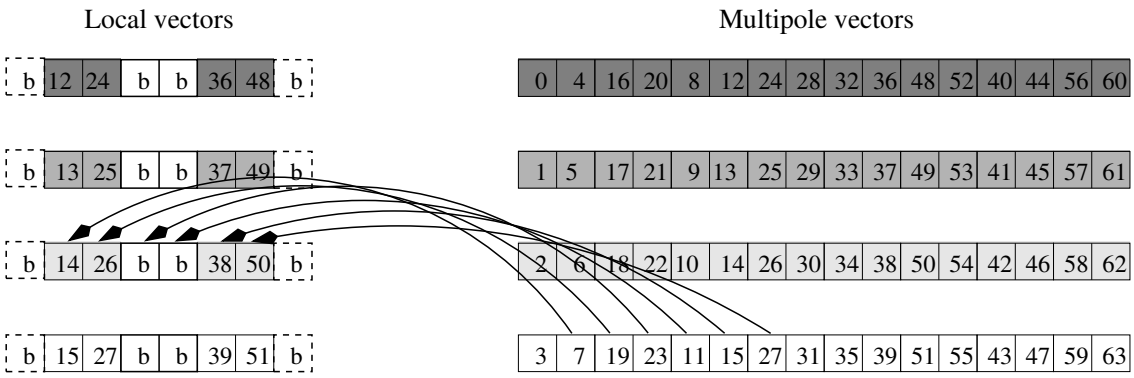


Figure 6: *slice* storage ($2D$, $ws = 1$, level 3): see also figure 5. The rows of the *row* data storage mode are concatenated with *blank boxes* (marked with 'b') in between: the matrix-matrix product now involves multipole and local matrices with 6 columns, against 2 columns with *row* storage (see figure 5).

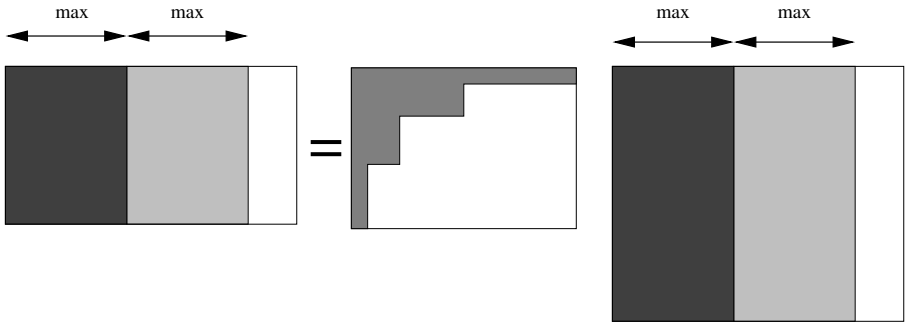


Figure 7: $NbExp_{max}$ decomposition (max denotes $NbExp_{max}$). This matrix-matrix product is computed as several matrix-matrix products with at most $NbExp_{max}$ columns in the multipole and local matrices.

Optimal values for $NbExp_{max}$ will be experimentally determined in section 3.3.1.

3.3 Tests and comparisons

Performance tests have been performed in order to validate the BLAS implementation, along with its parameters, that leads to the fastest $M2L$ computation. These tests have been performed either on one IBM Power3-II WH2+ (375 MHz, 1.5 GFLOPS, L1 cache size: 64 KB, L2 cache size: 4 MB, and 2 GB of memory) or on one IBM Power4+ (1454 MHz, $\approx$ 6 GFLOPS, L1 cache size: 64 KB, L2 cache size: 1.41 MB, L3 cache size: 32 MB, and 8 GB of memory), both at LaBRI[7]. The number of particles used for the simulation is not usually precised since the runtimes here measured depend only on the height of the unifom octree and on P, but of course the height used implies a range in the number of particles that balances the near field and the far field computations.

3.3.1 Decomposition for single height $M2L$ kernel

In this section, we will see how we have established, for each architecture (Power3 or Power4), the best $NbExp_{max}$ value, as well as the best s_{block} value for the *block_blas* routine (see section 3.1.2).

[7]Laboratoire Bordelais de Recherche en Informatique, Talence, FRANCE.

	$level = 4$	$level = 5$	$level = 6$	$level = 7$
recopies	216	2744	27000	238328
row storage	6	14	30	62
slice storage	46	222	958	3966

Table 1: $NbExp$ values according to our different schemes (with FBC, and $ws = 1$, see section 3.2.2).

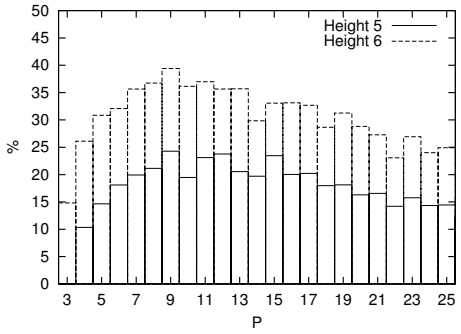
As use of level 3 BLAS calls always improves the performances over level 2 ones (see [15]), these values have been determined when using matrix-matrix products.

First we need to determine the optimal $NbExp_{max}$ values. These are clearly machine dependent, and they depend on P too since P determines the number of rows in the multipole matrix and in the local matrix. Depending on the level 3 BLAS scheme used and on the level in the octree, the number of multipole and local expansions concatenated, denoted by $NbExp$, differs significantly. For performance tests with different $NbExp_{max}$ values, we have only considered $NbExp$ values for cells with complete interaction list since this corresponds to the majority of the matrix-matrix products performed. The table 1 shows $NbExp$ values according to our different schemes for cells with complete interaction list.

In practice, tests for optimal $NbExp_{max}$ values will be performed, according to P, for one single "big enough" $NbExp$ value. Indeed, as confirmed by more complete tests, higher $NbExp$ values will have the same optimal $NbExp_{max}$ values and lower ones will not need such decomposition. Generally 2744 or 958 are sufficient: see for example figure 8. Moreover, results have shown that $NbExp_{max}$ is mainly independent from s_{block}: when splitting both multipole and local matrices according to $NbExp_{max}$, the sub-matrices size depends indeed only on P (for the number of rows) and $NbExp_{max}$ (for the number of columns): see figure 7.

Besides, the optimal s_{block} values, which depend on P and on the machine used, were searched between 1 and P. For low values of P, the best s_{block} values equal P, which means it is better to use only strips for the decomposition. But for higher values of P, the best s_{block} values were lower than P which justifies the recursive decomposition: see figure 8.

Finally, with these optimal s_{block} values, the relevance of the $NbExp_{max}$ decomposition is shown on figure 8 for the scheme with *recopies*. With growing octree height, the numbers of expansions $NbExp$ treated with one single matrix-matrix product increases, and the gain offered by the $NbExp_{max}$ decomposition increases thus too. With *row* and *slice* storages, $NbExp_{max}$ decomposition will be likewise relevant when the $NbExp$ value exceeds the optimal $NbExp_{max}$ value.



Gain in percentage.

P	s_{block}	$NbExp_{max}$	P	s_{block}	$NbExp_{max}$
3	3	921	15	5	102
4	4	936	16	3	96
5	5	463	17	3	84
6	5	403	18	3	84
7	5	307	19	4	102
8	3	260	20	7	126
9	7	210	21	3	120
10	2	168	22	2	120
11	10	156	23	5	132
12	10	126	24	6	120
13	1	138	25	11	120
14	1	120			

s_{block} and $NbExp_{max}$ values (Power4).

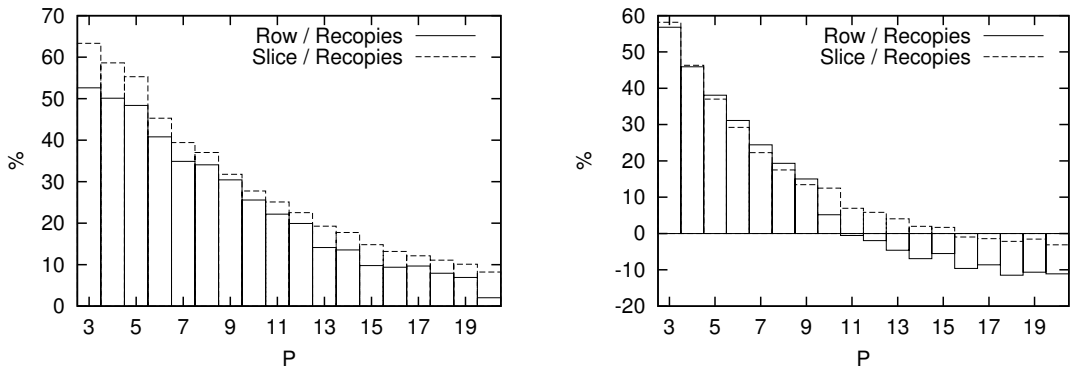
Figure 8: Gain in percentage for the full downward pass CPU times offered by a computation with $NbExp_{max}$ decomposition relative to one without $NbExp_{max}$ decomposition. Tests performed on IBM Power4, using the *block_blas* routine (with optimal s_{block}) and the scheme with *recopies* for an octree height equal to 5 ($NbExp = 2744$) and 6 ($NbExp = 27000$). The corresponding values for s_{block} and $NbExp_{max}$ are also given.

3.3.2 Recopies and special data storages

In order to compare the scheme with *recopies* with *row* and *slice* storages, CPU times have been measured on the full downward pass of the FMM. As mentioned in [21], the cost of copying vectors relatively to the cost of the matrix-matrix product decreases for growing values of P: copies of multipoles and local expansions are indeed performed in $\mathcal{O}(NbExp \times P^2)$ while the matrix-matrix product requires $\mathcal{O}(NbExp \times P^4)$ operations. This is more obvious with double height kernel than with single height kernel since the amount of computation for the matrix-matrix product is much more costly with double height, while the cost of *recopies* is the same. Figure 9 shows the gain of *row* and *slice* storages relative to the scheme with *recopies* according to different values of P for an octree height of 6: *row* and *slice* storages offer thus better gains relative to *recopies* for low values of P, and these gains are always higher for single height kernel.

These gains are also influenced by the height of the octree: with growing heights of the octree, the number of expansions treated with one matrix-matrix product increases for *row* and *slice* storages (see table 1) and the proportion of *blank boxes* decreases for *slice* storage. And for low heights of the octree, the use of *recopies* may sometimes be faster as in figure 9.

Slice storage is here more efficient than *row* storage because of the too small $NbExp$ for *row* storage at this height: for higher heights, *row* storage is a little bit faster. Moreover *slice* storage needs very long consecutive memory areas that cannot be allocated for too high values of P or too high heights when memory allocation with *row* storage may succeed. For these reasons we prefer *row* storage, and since the $NbExp$ values enabled by *slice* storage seem to be high enough, *level* storage has also been discarded.



(a) *block_blas* routine (single height kernel).

(b) *full_blas* routine (double height kernel).

Figure 9: Gain in percentage offered by *row* and *slice* storages relative to the scheme with *recopies* for downard pass CPU times with an octree of height 6. Tests performed on IBM Power4.

At last, as mentioned in sections 2.3 and 2.4, no BLAS can be applied to the FFT and Rotations improvements, and attempts to apply the scheme with *recopies* or *row* data storage have failed in improving the performances of these computations.

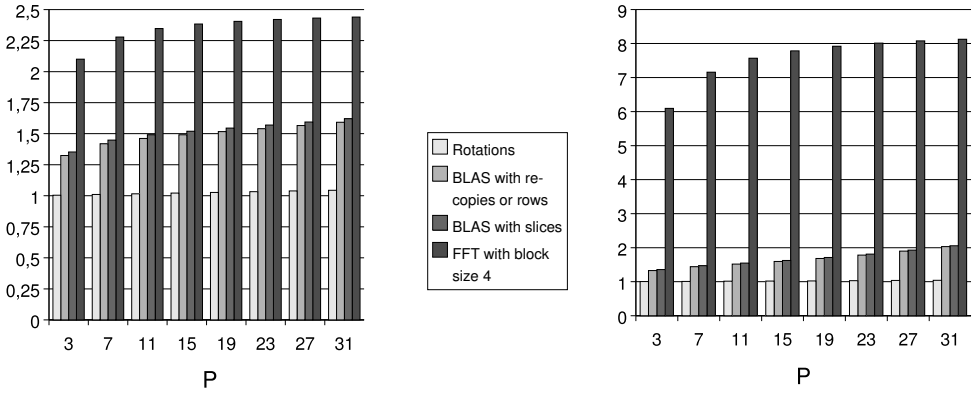
Now that we have determined the best BLAS implementations, namely *block_blas* and *full_blas* (depending on the $M2L$ kernel height), with *row* data storage, we are able to practically compare them with the other $M2L$ improvements.

4 Comparison of the $M2L$ improvements

4.1 Memory requirements

We first compare the memory requirements of the different schemes since it may be the first choice criterion. We here focus only on the memory used for the expansions: the memory used for the particles remains indeed unchanged when computing $M2L$ operator differently. In a nutshell, $M2L$ computation with rotations only needs extra memory for the precomputed $M2L$ transfer functions whose number is constant (316 with $ws = 1$). BLAS computation needs multipole vectors, whose size is roughly twice the size of the multipole expansions, and extra memory for the $M2L$ transfer matrices, especially with double height kernel since these matrices are dense in this case. *slice* data storage requires also extra memory for the *blank boxes*. At last $M2L$ computation with block or non-block FFT uses bigger arrays for its multipole expansions and its $M2L$ transfer functions in Fourier space: these arrays are roughly 4 times bigger with single height $M2L$ kernel and 16 times bigger with double height kernel. Using more detailed theoretical estimations that can be found in [15], we have plotted in figure 10, for each scheme, the ratio of its memory need to the classic $M2L$ memory need, using an octree of height 6 and only multiples of the FFT block size.

As predicted, while the rotation requirements are almost unnoticeable, and the BLAS ones remain moderate, FFT extra memory appear as problematic, especially for double height $M2L$ kernel. The same ratios apply for other FFT block sizes. Moreover the ratio of 2 for BLAS in the double height kernel with high values of P is due to the dense $M2L$ transfer matrices: for higher octree heights, this ratio remains close to 1.5 since the number of $M2L$ functions is constant while the number of cells in the octree grows exponentially. At last the additional cost of the *blank boxes* in *slice* storage over *row* storage is very small.



Single height

Double height

Figure 10: Memory requirements of the different $M2L$ computation schemes for an height of 6 (ratios to classic $M2L$ computation requirements).

4.2 CPU times for single height $M2L$ kernel

Figure 11 compares the different schemes for an octree of size 5. If the BLAS version outperforms the version with the classic $M2L$ and the one with rotations, the block FFT is faster, especially for values of P higher than 10. However in this test, without rescaling of the particle coordinates, the FFT with block size 4 is unstable for $P \geq 14$, and the one with block size 3 for $P \geq 23$.

When focusing on low and medium precisions, we recall also that for growing heights of the octree, the BLAS computation with *row* or *slice* storages becomes more efficient since the length of rows and slices increases at the leaf level which represents most of the runtime: the number of expansions treated in one level 3 BLAS call thus increases, as well as the efficiency of the BLAS. Figure 12 shows that BLAS is clearly competitive with the FFT with block size 4, in an octree

of height 7 and for low and medium precisions. Moreover the memory requirements of the FFT improvement are problematic here, since on figure 12, the brutal increase in runtime for FFT at $P = 10$ is due to swapping despite the 16 GB of memory available.

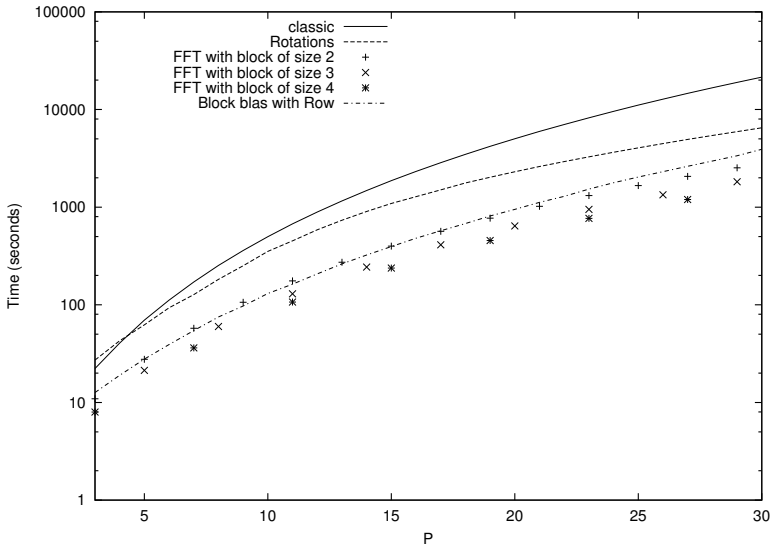


Figure 11: Logarithmic downward pass CPU times of the different $M2L$ computation schemes, for an octree of height 5 and with single height $M2L$ kernel. Tests are performed on an IBM Power3 with 2 GB of memory.

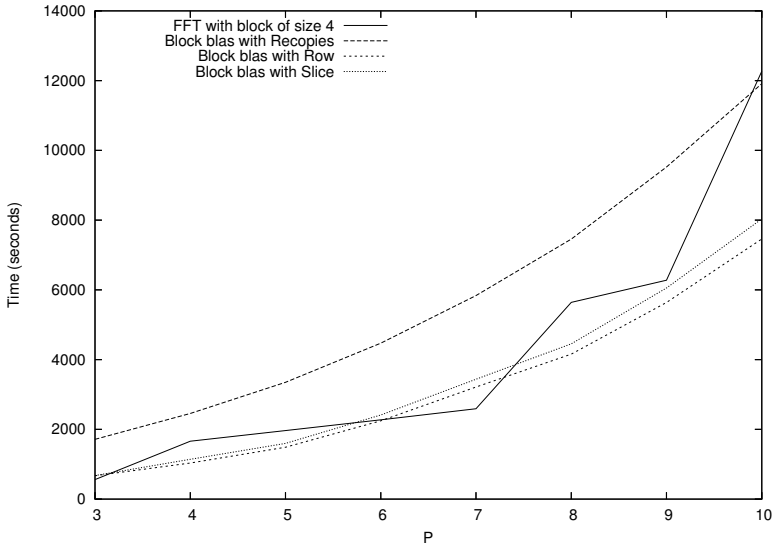


Figure 12: Downward pass CPU times of the different $M2L$ computation schemes, for an octree of height 7 and with single height $M2L$ kernel. Tests are performed on an IBM Power3 with 16 GB of memory at CINES (Centre Informatique National de l'Enseignement Supérieur, Montpellier, FRANCE).

4.3 CPU times for double height $M2L$ kernel

With double height kernel, the use of BLAS clearly outperforms all other methods. Figure 13 compares the different schemes for an octree of height 5. For the BLAS, we have plotted only the *row* data storage, but *slice* storage and the scheme with *recopies* have similar performances. For the FFT, the block version with blocks of size 4 have been plotted; in this test, it is only stable for $P \leq 8$, and more stable FFT with lower block sizes are slower. Moreover, we have stop the tests at $P = 19$ since on the IBM Power3, the 2GB of memory were insufficient for all FFT block sizes with $P > 19$. However it must be noticed that at this octree height, the rotation scheme becomes faster than all the BLAS schemes for $P \geq 23$: this is of course the aftermath of the lower operation count for the rotation scheme. Finally, as for single height kernel, the BLAS computation with *row* and *slice* storages become even more efficient for higher heights of the octree.

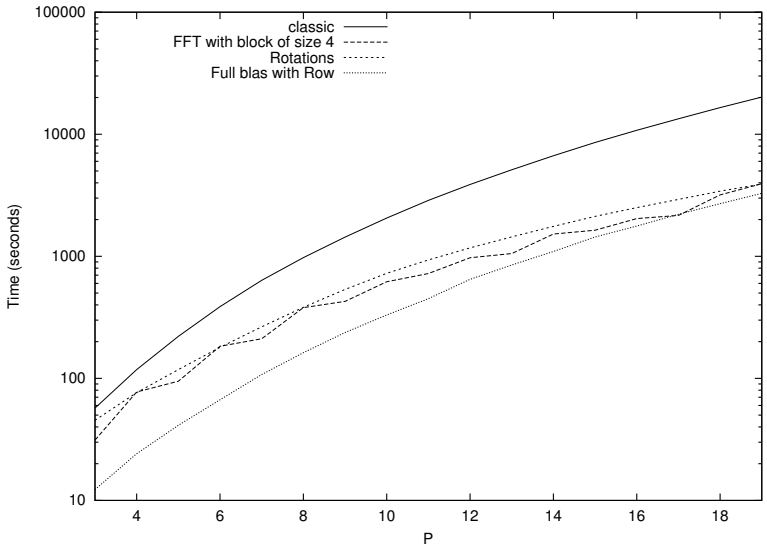


Figure 13: Logarithmic downward pass CPU times of the different $M2L$ computation schemes, for an octree of height 5 and with double height $M2L$ kernel. Tests are performed on an IBM Power3 with 2 GB of memory.

4.4 Computational efficiency

	Single height $M2L$ kernel		Double height $M2L$ kernel	
	$P = 7$	$P = 15$	$P = 7$	$P = 15$
classic	9.4 %	16.3 %	14.5 %	18.3 %
rotations	5.4 %	7.8 %	8.4 %	10.9 %
FFT with block size 4	4.5 %	13 %	12.4 %	18.4 %
level 3 BLAS	46.4 %	74 %	85.9 %	89.2 %

Table 2: Computational efficiencies of the different $M2L$ computation schemes.

In order to illustrate the efficiency of the BLAS versions that makes them faster than the other schemes, we present the table 2 that shows the MFLOPS (Millions of Floating Point Operations per Seconds) when computing $M2L$. The results are shown as percentages of the peak performance of the IBM Power3 used: 1500 MFLOPS. They have been computed with the Hardware Performance Monitor (HPM) Toolkit (version 2.4.3) as the average MFLOPS rate over all $M2L$ computations of a full FMM computation. For level 3 BLAS, we have used *block_blas* and *full_blas* routines

for respectively single and double height kernels, with an octree of height 5 and with *recopies* (the additional cost of copying the expansions is not considered here since we focus on the BLAS routine efficiency). It takes into account all cells, i.e. with and without complete interaction list. The height of the octree does not matter for classic $M2L$, rotations and FFT. These results can however not be directly compared from one row to the other: we recall here that the operations count differs! However it clearly illustrates how efficiently the processor is used in the different implementations and why BLAS computations outperform the other schemes.

This table also indicates that our decomposition of the matrix-matrix product for single height kernel in the *block_blas* routine, though satisfactory, is not optimal when compared to the *full_blas* routine efficiency of the double height kernel, especially for low values of P.

4.5 Single versus double height kernels

We have already seen (see theorem 4) that a sharp error bound has been theoretically proven only for the single $M2L$ height kernel. The double height $M2L$ kernel is certainly more precise, and for a given precision, it would therefore requires a lower value for P than the single height one. But since no precise error bound is available for double height kernel, we cannot a priori know what this lower P will be.

That is why we compare here CPU times with practical accuracies for both kernel heights. As already exposed (see for example [31] for double height $M2L$ kernel, and [27] for single height kernel), these practical accuracies are better than the theoretical ones which correspond to worst-case errors. These worst-case errors are indeed obtained with spherical regions, while we use in fact smaller cubical cells because of the octree. Moreover, we use relative error instead of absolute one so that the results are problem independent, and these relative errors are computed for the potential and not for the force components because these latters can be almost null. Since some discontinuities appear in the potential error for particles crossing cell boundaries (see [27]), we choose to study the more stable RMS average error instead of the maximum error. This RMS average error ε_{rms} is computed as follows:

$$\varepsilon_{rms} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\Phi_{Dir} - \Phi_{FMM}}{\Phi_{Dir}}\right)^2}.$$

We consider here a gravitational potential computed for an uniform distribution with 100000 bodies and an octree of height 4, which results in 25 bodies per leaf in the mean. We recall that when the number of bodies per leaf increases, the part of the near field (directly computed) in the potential becomes higher and the potential becomes thus more precise. Figure 14 presents, for each $M2L$ scheme, the tradeoff between practical error and CPU times (downward pass only) with both single and double height kernels. The scales are logarithmic, and the values of P plotted for double height kernel range from 3 to 14 with a step of 1, while for single height kernel they range from 3 to 29 with step 2: $P = 14$ in double height and $P = 29$ in single height correspond indeed to the same accuracy (precisely the first accuracy below 1.0×10^{-9} in our simulation).

As far as classic $M2L$ is concerned, the single height kernel is more efficient for low precisions and the double height one for high precisions. And as theoretically justified in sections 2.3, 2.4 and 4.4, while rotation and BLAS computations favor the double height kernel, the FFT improvement is generally more efficient with single height kernel.

Therefore we still have to compare the best of each scheme: this is done in figure 15 where we present FFT and BLAS versions on the same plot (classic and rotations have already been discarded in sections 4.2 and 4.3). The BLAS in double height kernel are then clearly more efficient than the FFT with single height kernel: by examples, for a practical error below 1.0×10^{-6} (respectively 1.0×10^{-8}) the gain is 35% (respectively 30%). This fully validates the relevance of our new BLAS version compared to the previous $M2L$ improvements.

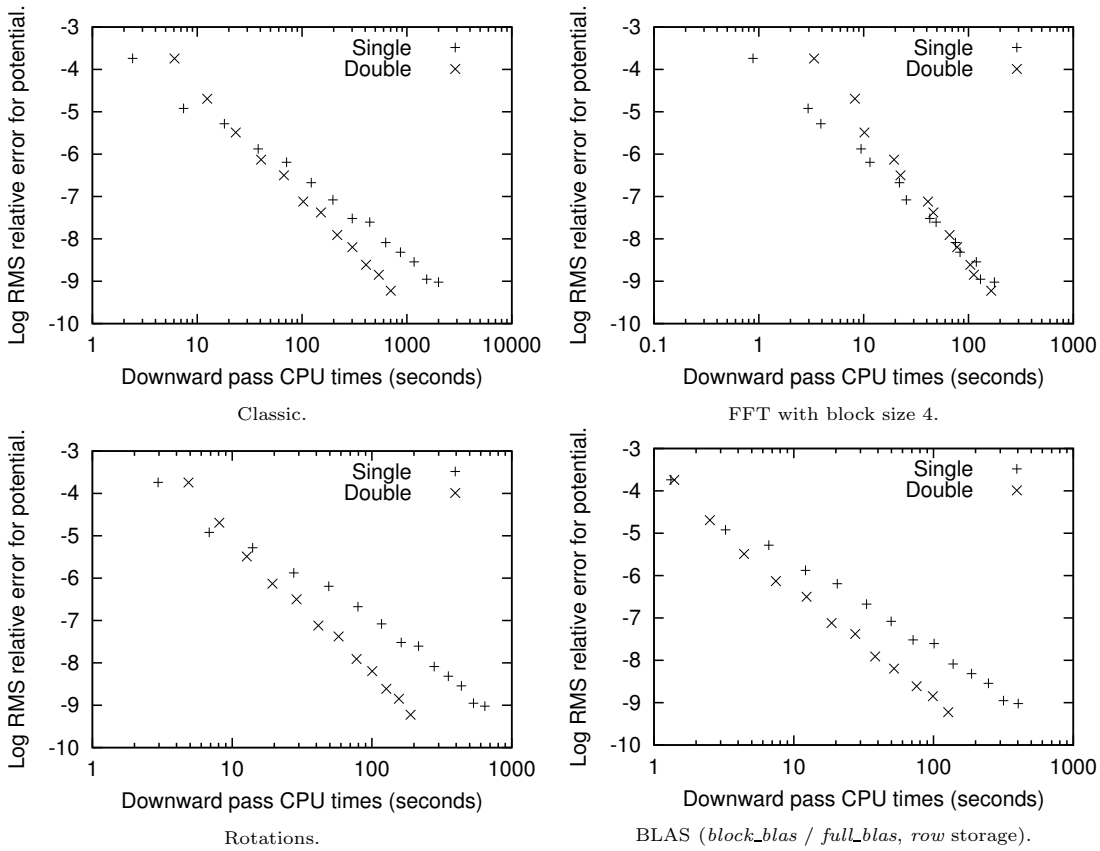


Figure 14: Tradeoffs between practical accuracies and CPU times for single and double height kernels.

5 Conclusion

In this paper, we have presented an overall study of the best implementation of the Fast Multipole Method for the serial computation of gravitational or electrostatic simulations of uniform distributions.

A detailed study of the error bound has lead us to two expressions of the $M2L$ operator that converts a multipole expansion into a local expansion: while the *double height $M2L$ kernel* is generally more efficient, only the *single height $M2L$ kernel* ensures a sharp error bound. For each $M2L$ expression, we have efficiently implemented the *block FFT* and the *rotation* improvements. To our knowledge, this is the first implementation of the FFT enhancement for double height $M2L$ kernel, and it has been shown that numerical instabilities remain even with the block FFT.

As an alternative, a BLAS (Basic Linear Algebra Subprograms) version has been proposed for the dense matrices of the double height kernel as well as for the sparse matrices of the single height one. A scheme with *recopies* has first made possible the use of level 3 BLAS resulting in impressive speedups. Special data storages for the expansions either by *rows* or by *slices* have then enabled us to avoid the additional cost of *recopies*, especially for low precisions.

Memory requirements estimations and CPU times from practical simulations have been used to precisely compare all these different schemes. It appears that the BLAS version and the FFT improvement with blocks are the most efficient ones. While the BLAS version is always faster in case of double height kernel, the block FFT is faster with single height kernel for high precisions. However the memory requirements of the block FFT as well as the remaining numerical instabilities, precisely for high precisions, limit severely the benefit it offers for runtime in this case. When comparing the tradeoff between practical accuracy and runtime for both $M2L$ kernel

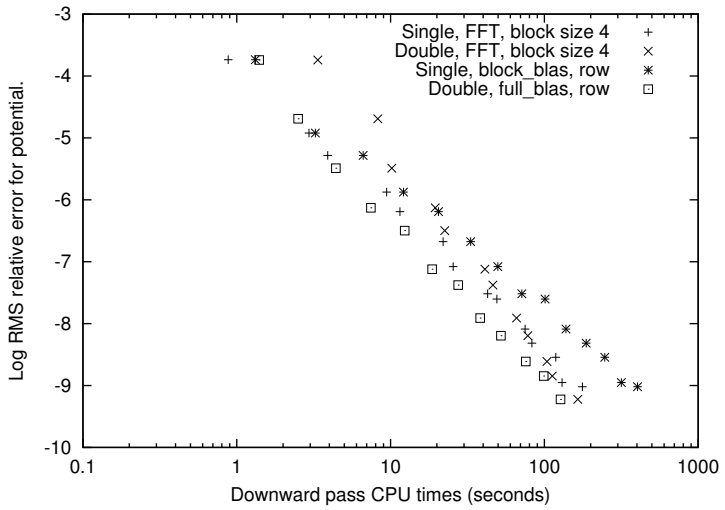


Figure 15: FFT block size 4 vs. BLAS (*block_blas* / *full_blas*, *row* storage).

heights, the BLAS with double height kernel is then always the most efficient, while introducing no numerical instabilities and low extra memory requirements.

In the future, we plan to apply BLAS to the adaptive version of the FMM (see [4] or [24]), and then to see how this will match its parallelization for both uniform and non uniform distributions on distributed memory architectures. This BLAS approach could also be extended to other potentials, whose FMM operators can be written as matrix products.

Acknowledgements

We are grateful to Guillaume Latu, ICPS-LSIIT (Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection) and INRIA CALVI project, for several helpful discussions on the underlying techniques of BLAS and their integration in the FMM.

References

[1] *Fortran 77 reference implementation source code of the blas from netlib.* Electronically available at: `http://www.netlib.org/blas`.

[2] C. R. Anderson, *An implementation of the fast multipole method without multipoles*, SIAM Journal on Scientific and Statistical Computing, 13 (1992), pp. 923–947.

[3] J. C. Carr, R. K. Beatson, J. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans, *Reconstruction and representation of 3D objects with radial basis functions*, in ACM SIGGRAPH 2001, Los Angeles, CA, August 2001, pp. 67–76.

[4] J. Carrier, L. Greengard, and V. Rokhlin, *A fast adaptive multipole algorithm for particle simulations*, SIAM Journal on Scientific and Statistical Computing, 9 (1988), pp. 669–686.

[5] H. Cheng, L. Greengard, and V. Rokhlin, *A fast adaptive multipole algorithm in three dimensions*, Journal of Computational Physics, 155 (1999), pp. 468–498.

[6] C. H. Choi, J. Ivanic, M. S. Gordon, and K. Ruedenberg, *Rapid and stable determination of rotation matrices between spherical harmonics by direct recursion*, Journal of Chemical Physics, 111 (1999), pp. 8825–8831.

[7] E. Darve, *Méthodes multipôles rapides : résolution des équations de Maxwell par formulations intégrales*, PhD thesis, Université Paris 6, 1999.

[8] E. Darve and P. Havé, *Efficient fast multipole method for low-frequency scattering*, Journal of Computational Physics, 197 (2004), pp. 341–363.

[9] J. J. Dongarra, J. D. Croz, S. Hammarling, and I. Duff, *A set of level 3 Basic Linear Algebra Subprograms*, ACM Transactions on Mathematical Software, 16 (1990), pp. 1–17.

[10] J. J. Dongarra, J. D. Croz, S. Hammarling, and R. J. Hanson, *An extended set of FORTRAN Basic Linear Algebra Subprograms*, ACM Transactions on Mathematical Software, 14 (1988), pp. 1–17.

[11] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst, *Numerical Linear Algebra for High-Performance Computers (Software, Environments, Tools)*, SIAM, Philadelphia, PA, USA, 1998.

[12] W. Elliott, *Multipole algorithms for molecular dynamics simulation on high performance computers*, Tech. Rep. 95-003, Duke University, Department of Electrical Engineering, 1995. Doctoral dissertation.

[13] W. D. Elliott and J. A. Board, Jr., *Fast Fourier Transform accelerated fast multipole algorithm*, SIAM Journal on Scientific Computing, 17 (1996), pp. 398–415.

[14] M. A. Epton and B. Dembart, *Multipole translation theory for the three-dimensional Laplace and Helmholtz equations*, SIAM Journal on Scientific Computing, 16 (1995), pp. 865–897.

[15] P. Fortin, *Multipole-to-local operator in the Fast Multipole Method: comparison of FFT, rotations and BLAS improvements*, Research Report 5752, INRIA, 2005.

[16] M. Frigo and S. G. Johnson, *The design and implementation of FFTW3*, Proceedings of the IEEE, 93 (2005), pp. 216–231. special issue on "Program Generation, Optimization, and Platform Adaptation".

[17] L. Greengard, *The Rapid Evaluation of Potential Fields in Particle Systems*, MIT Press, Cambridge, MA, 1988.

[18] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, Journal of Computational Physics, 73 (1987), pp. 325–348.

[19] ———, *A new version of the fast multipole method for the Laplace equation in three dimensions*, Acta Numerica, 6 (1997), pp. 229–269.

[20] N. A. Gumerov and R. Duraiswami, *Recursions for the computation of multipole translation and rotation coefficients for the 3-D Helmholtz equation*, SIAM Journal on Scientific Computing, 25 (2003), pp. 1344–1381.

[21] Y. Hu and S. L. Johnsson, *Implementing O(N) N-body algorithms efficiently in data-parallel languages*, Scientific Programming, 5 (1996), pp. 337–364.

[22] B. Kågström, P. Ling, and C. van Loan, *GEMM-based level 3 BLAS: high-performance model implementations and performance evaluation benchmark*, ACM Trans. Math. Softw., 24 (1998), pp. 268–302.

[23] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, *Basic Linear Algebra Subprograms for FORTRAN usage*, ACM Transactions on Mathematical Software, 5 (1979), pp. 308–323.

[24] K. Nabors, F. T. Korsmeyer, F. T. Leighton, and J. White, *Preconditioned, adaptive, multipole-accelerated iterative methods for three-dimensional first-kind integral equations of potential theory*, SIAM Journal on Scientific Computing, 15 (1994), pp. 713–735.

[25] K. Nabors and J. White, *Fastcap: A multipole accelerated 3-D capacitance extraction program*, IEEE Transactions on Computer-Aided Design, 10 (1991), pp. 1447–1459.

[26] H. Petersen, E. Smith, and D. Soelvason, *Error estimates for the fast multipole method. II. The three-dimensional case*, Proceedings of the Royal Society of London. Series A, Mathematical and physical sciences, 448 (1995), pp. 401–418.

[27] W. Rankin, *Efficient parallel implementations of multipole based N-body algorithms*, PhD. Dissertation, Duke University, Department of Electrical Engineering, April 1999.

[28] J. P. Singh, C. Holt, T. Totsuka, A. Gupta, and J. Hennessy, *Load balancing and data locality in adaptive hierarchical N-body methods: Barnes-Hut, fast multipole, and radiosity*, Journal of Parallel and Distributed Computing, 27 (1995), pp. 118–141.

[29] D. Soelvason and H. Petersen, *Error estimates for the fast multipole method*, Journal of Statistical Physics, 86 (1997), pp. 391–420.

[30] X. Sun and N. P. Pitsianis, *A matrix version of the fast multipole method*, SIAM Review, 43 (2001), pp. 289–300.

[31] C. A. White and M. Head-Gordon, *Derivation and efficient implementation of the fast multipole method*, Journal of Chemical Physics, 101 (1994), pp. 6593–6605.

[32] C. A. White and M. Head-Gordon, *Rotating around the quartic angular momentum barrier in fast multipole method calculations*, Journal of Chemical Physics, 105 (1996), pp. 5061–5067.