Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 108C (2017) 1743–1752

Procedia Computer Science

International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland

Solution of Few-Body Coulomb Problems with Latent Matrices on Multicore Processors

Luis Biedma[1,2], Flavio Colavecchia[2,3], and Enrique S. Quintana-Ortí[4]

[1] FAMAF, Universidad Nacional de Córdoba, Argentina
[2] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Argentina
[3] Div. Física Atómica, Molecular y Óptica, Centro Atómico Bariloche, Argentina
[4] Depto. Ingeniería y Ciencia de Computadores, Universidad Jaume I (UJI), Castellón, Spain

Abstract

We re-formulate a classical numerical method for the solution of systems of linear equations to tackle problems with *latent* data, that is, linear systems of dimension that is a priori unknown. This type of systems appears in the solution of few-body Coulomb problems for Atomic Simulation Physics, in the form of multidimensional partial differential equations (PDEs) that require the numerical solution of a sequence of recurrent dense linear systems of growing scale. The large dimension of these systems, with up to several hundred thousands of unknowns, is tackled in our approach via a task-parallel implementation of a solver based on the QR factorization. This method is parallelized using the OmpSs framework, showing fair strong and weak scalability on a multicore processor equipped with 12 Intel cores.

Peer-review under responsibility of the scientific committee of the International Conference on Computational Science

Keywords: Few-body problems, latent matrices, QR factorization, task-parallelism, multicore CPUs

1 Introduction

A key method to solve many of the differential equations that describe the physical world is to expand the solution in a basis of functions. This transforms the problem of obtaining the solution of the differential equation into the calculation of the coefficients of its expansion. Usually the basis is mathematically defined in an infinite-dimensional vector space. The physicist thus faces the challenge of to compute the relevant magnitudes of the physical system under scrutiny, choosing a basis large enough such that the calculation is meaningful, with the restrictions of memory and computer power at hand. Moreover, the optimal dimension of the basis set is usually not known *a priori*, and many trial-and-error runs need to be performed.

To establish the problem in detail, let us consider a differential operator $\hat{A}(r)$ in an infinite-dimensional vector space given by the variables $r = \{r_1, r_2, \ldots, r_n\}$, and a general non-homogeneous partial differential equation (PDE) in that space given by:

$$\hat{A}(r)\Psi(r) = \Psi_0(r). \tag{1}$$

1877-0509
Peer-review under responsibility of the scientific committee of the International Conference on Computational Science
10.1016/j.procs.2017.05.019

For this problem, we are interested in the solution $\Psi(r)$ for a given set of boundary conditions. Consider an orthonormal basis set given by the functions $\{\phi_n(r)\}$ such that the internal product is $< \phi_n, \phi_m >= \delta_{nm}$. Thus, the functions $\Psi(r)$ and $\Psi_0(r)$ in this basis can be expanded as:

$$\Psi(r) = \sum_n x_n \phi_n(r), \qquad \Psi_0(r) = \sum_n b_n \phi_n(r), \tag{2}$$

which include all the infinite elements of the basis. To make use of these expressions in a computer application and obtain a numerical solution to equation (1), it is necessary to select only N elements of the basis, such that

$$\Psi(r) \approx \Psi^{(N)}(r) = \sum_{n=1}^{N} x_n \phi_n(r), \qquad \Psi_0(r) \approx \Psi_0^{(N)}(r) = \sum_{n=1}^{N} b_n \phi_n(r). \tag{3}$$

In this way, equation (1) is projected into the selected truncated basis, obtaining a linear system for the coefficients $x^{(N)} = \{x_1, x_2, \ldots, x_N\}$:

$$A^{(N)} x^{(N)} = b^{(N)}, \tag{4}$$

where the matrix $A^{(N)}$, of size $N \times N$, has elements $A^{(N)}_{nm} =< \phi_n, \hat{A}\phi_m >$; and the vector $b^{(N)} = \{b_1, b_2, \ldots, b_N\}$ is the approximate representation of the initial condition Ψ_0 in the truncated basis.

The usual work-flow to obtain the best solution $\Psi(r)$ is to solve the system (4) for increasing values $N_1 < N_2 < N_3 \ldots$ of N until an adequate convergence is achieved. This convergence and the associated stopping criterion are usually determined by the precision required in the calculation of a physical magnitude, which is bounded by experimental data.

This naive procedure presents several drawbacks. First, one can choose to compute the complete matrix $A^{(N)}$ for each $N_1 < N_2 < N_3 \ldots$, with the result that many elements of the matrix are repeatedly computed as all the elements of $A^{(N_i)}$ are included for all the calculations with $N \geq N_i$, for $i = 1, 2, \ldots$. To avoid these costly recalculations, the different matrices $A^{(N_i)}$ can be saved, but then that data will have to be retrieved from files, since memory constraints are reached very rapidly for medium-sized matrices. In this last scenario, the calculation is limited by the slow I/O transfer rate from disk. Second, although numerical linear algebra packages provide the functionality to tackle a substantial amount of problems, they all need to know in advance the size, structure and data of the matrices in any of their many implementations: serial, multi-threaded and/or distributed algorithms.

In this work, we introduce a new concept of matrices, that we qualify as *latent*. The main features of these matrices are: 1) The dimension of the latent matrix is not known *a priori*; 2) a latent matrix has a well-known structure of blocks or tiles; 3) each element of a latent matrix can be computed with a well-defined numerical algorithm; and 4) the latency of the computation of each element can be determined, for a given computational setup.

The concept of latent matrices is useful to classify the problem stated in the preceding paragraphs. However, it raises the question of designing adequate numerical linear algebra algorithms and software for them. In this work we move towards the exploration of these algorithms for the solution of few-body Coulomb problems, making the following specific contributions:

- We re-formulate the solution of a system of linear equations with latent data into a matrix factorization updating problem, which can be performed via a "lazy" QR factorization.
- We analyze the parallelism of the problem, proposing an asynchronous solution that overlaps the execution of multiple updating problems/QR factorizations to expose a higher degree of task concurrency.

- We tackle the large-scale of the linear systems appearing in few-body Coulomb problems via a task-parallel implementation based on the OmpSs programming model.
- We evaluate the performance of our task-parallel solution for latent linear systems on a 12-core Intel-based server, showing reasonable scalability.

2 Physics Problems with Latent Matrices

One of the most important examples of latent matrices arises in Atomic Collisions Physics. This field is devoted to the calculation of collisions processes between photons, electrons, ions, atoms and molecules, which are present in a variety of applications, ranging from energy production from plasma reactors to the study of atmospheric reactions.

For example, the simplest atomic collision is the elastic scattering of an electron from a Hydrogen nucleus. In this process, two charged particles interact with each other following the rules of the long-range Coulomb interaction within the quantum world. Adding one particle to that system leads to a six-dimensional PDE which is analytically unsolvable. The most accurate framework to deal with this physical system relies on the numerical solution of the Schrödinger equation for the few-body problem that represents the collision. This has been the method of choice for rather small, but very complex systems at the atomic scale. Even nowadays, the problem is still extremely difficult to solve.

Let us consider the Schrödinger equation for continuum states that results from collisions in a three-body system, and can be obtained from the solution of the time-independent problem

$$(H - E)\,\Psi = W\Psi_0, \tag{5}$$

where Ψ_0 is the initial state of the process and E is the total energy of the system. The (possibly differential) operator W is responsible for the transitions from the initial, known state Ψ_0 to the continuum, collisional unknown state Ψ. For the case of two electrons and an atomic nucleus, one considers a coordinate system given by the vectors $\{\mathbf{r}_1, \mathbf{r}_2\}$, which represent the position of each electron with respect to the nucleus. Therefore, equation (5) is a six-dimensional PDE. The approach to solve this problem numerically is to use a spectral method, where one introduces a suitable basis set to represent the solution Ψ. One of the main features of this procedure is that the physical considerations can be introduced directly into the basis elements [1, 2, 3].

Any spectral method in the electronic coordinates introduces a basis set $\{\Xi^L_{nm}(\mathbf{r}_1, \mathbf{r}_2)\}$ to expand the wave function Ψ:

$$\Psi(\mathbf{r}_1, \mathbf{r}_2) = \sum_{nm}^{NM} x_{nm}\Xi^L_{nm}(\mathbf{r}_1, \mathbf{r}_2).$$

This represents the application of equation (2) for this particular atomic system. At this point we need to point out that the equation implicitly depends on the total angular momentum L. This fact is represented by the superscript in the definition of the basis. The Hamiltonian is a latent matrix in this basis set, and can be computed as:

$$\mathbf{H}^L_{n'm'nm} = \left\langle \Xi^L_{n'm'} \left| H - E \right| \Xi^L_{nm} \right\rangle,$$

which is a square matrix of $N_H \times N_H$ elements.

The solution of the linear system $\mathbf{Hx} = \mathbf{b}$ gives the set of coefficients $\mathbf{x} = \{x_{nm}\}$ that determines the solution Ψ. The right-hand side vector of this linear system, $\mathbf{b}$, is the projection into the basis of the action of operator W on the initial state Ψ_0.

The elements of the Hamiltonian matrix $\mathbf{H}^L_{n'm'nm}$ are a priori six-dimensional integrals in the coordinate space spanned by the set $\{\mathbf{r}_1, \mathbf{r}_2\}$. However, the complexity of the calculation of these elements can be reduced if physical symmetries are taken into account when the basis elements are selected. Let us assume that we make use of spherical coordinates for each electron: $\mathbf{r}_i = (r_i, \theta_i, \varphi_i)$ for $i = 1, 2$. With this choice, each element of the basis Ξ^L_{nm} is defined as a product of two one-electron functions [4]:

$$\Xi^L_{n_a n_b}(\mathbf{r}_1, \mathbf{r}_2) = \frac{S_{n_a l_a}(r_1)}{r_1} \frac{S_{n_b l_b}(r_2)}{r_2} \mathcal{Y}^{LM}_{l_a l_b}(\widehat{\mathbf{r}}_1, \widehat{\mathbf{r}}_2),$$

where $1 \leq n_a \leq N_a$ and $1 \leq n_b \leq N_b$, and $\{N_a, N_b\}$ are the number of elements for each one-electron basis set. The indexes l_a and l_b represent the angular momentum of each electron, and satisfy $|L - l_a| \leq l_b \leq L + l_a$. This implies that, for a given L, there are several pairs of values (l_a, l_b) that can be included in the basis.

From the computational point of view, this problem is *latent* in the basis size parameters $\{N_a, N_b\}$ as well as in the number of pairs (l_a, l_b). This means that the optimal values of these parameters are not known a priori, and usually a trial-and-error procedure is performed until the required accuracy is achieved.

Typically, several pairs of angular momenta are used for a given L (for example, from 6 to 16 [5]), and the basis size $N_a = N_b$ includes several dozens of elements. Therefore, the size of the linear system to be solved can feature several hundred thousands of unknowns. To conclude this section, the Hamiltonian is a dense matrix with complex entries that needs to be computed in double-precision arithmetic, raising the memory requirements to hundreds of Gbytes [6].

3 Linear Algebra Methods for Latent Linear Systems

We next describe in detail the linear algebra problem underlying the few-body Coulomb problems, connecting the linear systems with latent data to the updating of a dense QR factorization.

Solution of latent linear systems. The discussion in Section 2 exposes that, in order to tackle the target problem, we need to solve a sequence of linear systems of the form $\mathbf{Hx} = \mathbf{b}$, organized in a cycle of "levels". Thus, we generate and solve the problem at level s; next, we test whether the solution obtained at that level is satisfactory for the global problem; and in case it is not, we proceed to the next level $s + 1$, repeating the cycle (trial-and-error).

Let us define the process formally. Assume the linear system to be solved initially (i.e, at level 0) is given by

$$A^{(0)} x^{(0)} = b^{(0)}, \tag{6}$$

where $A^{(0)}$ is the $m_0 \times m_0$ coefficient matrix, $b^{(0)}$ is the right-hand side vector, $x^{(0)}$ is the sought-after solution, and $b^{(0)}, x^{(0)}$ are both of size m_0. In case the solution of (6) is not satisfactory, we then extend the linear system to

$$A^{(1)} x^{(1)} = b^{(1)} \equiv \begin{bmatrix} A^{(0)} & B^{(0)} \\ C^{(0)} & D^{(0)} \end{bmatrix} \begin{bmatrix} y^{(0)} \\ z^{(0)} \end{bmatrix} = \begin{bmatrix} b^{(0)} \\ c^{(0)} \end{bmatrix}, \tag{7}$$

where $A^{(1)}$ is $n_1 \times n_1$, $n_1 = m_0 + m_1$, and the components of (7) are partitioned accordingly.

This dependence between levels 0 and 1 can be generalized to two consecutive levels, $s-1$ and s, in the form of the algorithm for the solution of *recurrent latent linear systems* (RLLS) stated in Figure 1. The dense linear system in Step 2 there, corresponding to equation (6), can be solved via any conventional factorization algorithm, such as the LU or QR factorizations [7], for

1.	Generate the initial problem data: $A^{(0)}, b^{(0)}$.
2.	Solve the $m_0 \times m_0$ dense linear system $A^{(0)}x^{(0)} = b^{(0)}$.
3.	Solve the sequence of levels:
	for $s = 1, 2, \ldots$ until convergence
3.1	Generate the problem data for the s-th level:
	$B^{(s-1)}$ of size $n_{s-1} \times m_s$, $C^{(s-1)}$ of size $m_s \times n_{s-1}$, $D^{(s-1)}$ of size $m_s \times m_s$,
	where $n_{s-1} = \sum_{k=0}^{s-1} m_k$; and $c^{(s-1)}$ with m_s components.
3.2	Solve the $n_s \times n_s$ linear system for the s-th level:
	$A^{(s)}x^{(s)} = b^{(s)} \equiv \begin{bmatrix} A^{(s-1)} & B^{(s-1)} \\ C^{(s-1)} & D^{(s-1)} \end{bmatrix} \begin{bmatrix} y^{(s-1)} \\ z^{(s-1)} \end{bmatrix} = \begin{bmatrix} b^{(s-1)} \\ c^{(s-1)} \end{bmatrix}$.
	end for

Figure 1: Algorithm RLLS.

a cost of $2m_0^3/3$ or $4m_0^3/3$ floating-point arithmetic operations (flops), respectively. (Hereafter, we neglect minor order terms in the flop expressions such as, e.g., the cost required to solve the triangular systems after the matrix factorization. Furthermore, for simplicity, we assume real arithmetic though the actual calculations for a few-body problem operate with complex data.)

The solution of the linear system in Step 3.2 is more challenging. Concretely, assuming we rely on the QR factorization, a naive solution that tackles Step 3.2 in isolation will require $4n_s^3/3$ flops, with this figure rapidly growing with the level index s and the dimension of the subproblems m_j, $j = 0, 1, 2, \ldots$.

Updating a QR factorization. A more elaborate solution can avoid the large overhead of the previous naive approach by exploiting that $A^{(s-1)}$ is a leading principle submatrix of $A^{(s)}$ to leverage an existing factorization of the former. In linear algebra, this is known as an *updating problem*, and efficient high performance solutions have been proposed for the (incremental) QR factorization [8] and the LU factorization (with incremental pivoting) [9]. Due to the potential instability of the incremental pivoting technique required in the latter, in the remainder of this work we focus on the numerically-stable incremental QR factorization, reviewing the underlying procedure next.

Assume we have already computed the factorization $A^{(s-1)} = Q^{(s-1)}R^{(s-1)} = QR$, and consider the partitioning

$$\left[\begin{array}{c|c} Q^T A^{(s-1)} & B^{(s-1)} \\ \hline C^{(s-1)} & D^{(s-1)} \end{array}\right] = \left[\begin{array}{c|c} R^{(s-1)} & B^{(s-1)} \\ \hline C^{(s-1)} & D^{(s-1)} \end{array}\right] = \left[\begin{array}{cccc|c} R_{00} & R_{01} & \ldots & R_{0,s-1} & B_0 \\ & R_{11} & \ldots & R_{1,s-1} & B_1 \\ & & \ddots & \vdots & \vdots \\ & & & R_{s-1,s-1} & B_{s-1} \\ \hline C_0 & C_1 & \ldots & C_{s-1} & D \end{array}\right], \quad (8)$$

where the tiles on the diagonal $R_{k,k}$, of dimension $m_k \times m_k$, $k = 1, 2, \ldots, s-1$, are all upper triangular. The *structure-aware* procedure to update this matrix factorization (UMF) is then illustrated in Figure 2. Upon completion, the coefficient matrix (8) has been reduced to upper triangular form $R^{(s)}$ via a sequence of orthogonal transformations represented by $Q^{(s)}$. Therefore, by applying the same sequence of transformations to $b^{(s)}$, we can next use backward substitution to solve $R^{(s)}x^{(s)} = ((Q^{(s)})^T b^{(s)})$ for $x^{(s)}$.

The integration of the update procedure UMF into algorithm RLLS produces a lazy variant of the QR factorization where, at each step of the decomposition (level of the RLLS solver), we

1.	Apply previous transformations to $B^{(s-1)}$:	Cost (in flops):
	for $j = 0, 1, \ldots, s-1$	
1.1	$B_j := Q_{j,j}^T B_j$	$2m_j^3$
	for $i = j+1, j+2, \ldots, s-1$	
1.2	$\begin{bmatrix} B_j \\ \hline B_i \end{bmatrix} := Q_{i,j}^T \begin{bmatrix} B_j \\ \hline B_i \end{bmatrix}$,	$4((m_j + m_i)m_j^2 - m_j^3/2)$
	end for	
	end for	
2.	Factorize new row block in $[C \mid D]$:	
	for $j = 0, 1, \ldots, s-1$	
2.1	Compute the QR fact. $\begin{bmatrix} R_{j,j} \\ \hline C_j \end{bmatrix} = Q_{s,j} \begin{bmatrix} \bar{R}_{s,j} \\ \hline 0 \end{bmatrix}$ and set $R_{j,j} := \bar{R}_{s,j}$.	$2(m_s m_j^2 + 4m_j^3/3)$
	for $p = j+1, j+2, \ldots, s-1$	
2.2	$\begin{bmatrix} R_{j,p} \\ \hline C_p \end{bmatrix} := Q_{s,j}^T \begin{bmatrix} R_{j,p} \\ \hline C_p \end{bmatrix}$.	$4((m_j + m_s)m_j^2 - m_j^3/2)$
	end for	
2.3	$\begin{bmatrix} B_j \\ \hline D \end{bmatrix} := Q_{s,j}^T \begin{bmatrix} B_j \\ \hline D \end{bmatrix}$.	$4((m_j + m_s)m_j^2 - m_j^3/2)$
	end for	
3.	Compute the QR fact. $D = Q_{s,s}\bar{R}_{s,s}$, and set $D := \bar{R}_{s,s}$.	$4m_s^3/3$

Figure 2: Structure-aware procedure UMF (application at step s of algorithm RLLS). Here $Q_{00} = Q^{(0)}$ denotes the orthogonal factor resulting from the initial factorization of $A^{(0)}$ performed as part of Step 2 of algorithm RLLS.

only update the elements on the s-th column and s-th row of the matrix above and to the left of the (s, s) element, respectively. For performance, this is formulated as a blocked algorithm.

Exploiting the upper triangular structure of $A^{(s-1)}$ produces a considerable reduction in the cost with respect to the naive approach; see the column for the costs in Figure 2. Furthermore, due to the use of orthogonal transformations, the structure-aware procedure is stable [7].

4 Parallel Tools for the Solution of Latent Systems

The solution of conventional (i.e. non-latent) dense linear systems, on a general-purpose multicore processor, can be performed using the appropriate computational kernels (routines) from multi-threaded dense linear algebra libraries such as Intel MKL, PLASMA [10], or `libflame` [11]. Concretely, the parallel routines in these libraries can be easily applied to the solution of the initial system in (6). For the recurrence defining the latent systems, they can also be leveraged to compute in parallel the individual computational kernels composing the structure-aware procedure UMF and the subsequent solve (see Figure 2). However, this approach is constrained in the amount of the parallelism that is exploited, which does not lead to a scalable solution.

In this section we discuss an alternative strategy that exposes additional task-parallelism in the recurrence of latent linear systems, improving the scalability of the complete process. We open the section with a detailed analysis of the task-parallelism of this process. Then, we offer a glimpse of the OmpSs programming model that underlies our parallel solution.

Task-parallel solution of latent systems. Consider the structure-aware procedure UMF in Figure 2 applied at level $s = 4$. Let us denote each one of the factorizations in Step 2.1 there by F_j, $j = 0, \ldots, 3$; and the application of the orthogonal transformations comprised by $Q_{s,j}$ to each one of the submatrices in Step 2.2, by $\mathsf{U}_{j,p}$, $p = j+1, j+2, \ldots, 3$. Furthermore, for simplicity, let us skip the application of the transformations in Step 2.3 as well as the operations

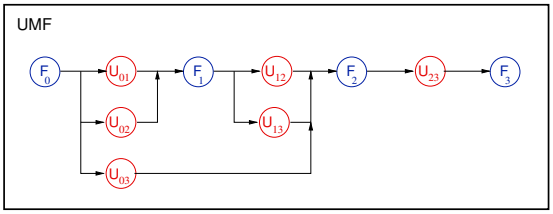

Figure 3: (Simplified) TDG for the structure-aware procedure UMF applied to matrix $A^{(s)}$ at level $s = 4$.

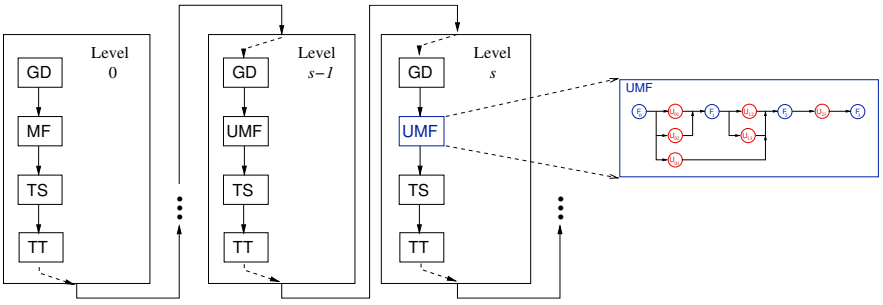

Figure 4: (Simplified) TDG for algorithm RLLS.

in Step 1 during the following discussion.

Figure 3 represents the data dependencies appearing in the matrix factorization update in the form of a task dependency graph (TDG), where the nodes identify the operations (tasks or computational kernels) and the edges specify the dependencies. This plot reports the concurrency available in this step of the algorithm as that present solely in the updates $\mathsf{U}_{j,p}$ for each independent value of p. Additional parallelism can be exposed by dividing $C^{(s-1)}/B^{(s-1)}$ in (8) into finer-grain block columns/rows of width/height $t_d < m_j$. (Note that this induces a partitioning of $R^{(s-1)}$ that presents diagonal blocks of dimension $t_d \times t_d$.) However, as t_d diminishes, the update of these blocks becomes a memory-bound kernel, offering very low performance.

An appealing opportunity to expose ampler task-parallelism comes from considering the solution of the full recurrence of latent linear systems. In particular, Figure 4 depicts the levels and steps appearing in the solution process (see Figure 1), using the following notation:

- GD: data generation (Steps 1 and 3.1);
- MF: matrix factorization (Step 2, only in level 0) and UMF for the update version (Step 3.2);
- TS: application of orthogonal transformations to the right-hand side and solution of the subsequent upper triangular system (following MF in level 0 or UMF in levels 1,2,...); and
- TT: termination test (stopping criterion).

For GD, it is possible to partition the problem data into multiple independent blocks, yielding an embarrassingly-parallel execution of this step. In contrast, TS involves operations with a minor cost and low parallel performance. Unfortunately, this organization of algorithm RLLS presents data dependencies between each pair of consecutive steps, and a control dependency (for the stopping criterion) between each pair of consecutive levels, that introduce a synchronization

point each:

$$\overbrace{\mathsf{GD} \rightarrow \mathsf{MF} \rightarrow \mathsf{TS} \rightarrow \mathsf{TT}}^{\text{Level } 0} \rightarrow \cdots \rightarrow \overbrace{\mathsf{GD} \rightarrow \mathsf{UMF} \rightarrow \mathsf{TS} \rightarrow \mathsf{TT}}^{\text{Level } s-1} \rightarrow \overbrace{\mathsf{GD} \rightarrow \mathsf{UMF} \rightarrow \mathsf{TS} \rightarrow \mathsf{TT}}^{\text{Level } s} \rightarrow \cdots$$

The approach we propose here is to merge GD with MF (in level 0) or UMF (in levels 1 and higher) so that the first task accessing a block of $C^{(s-1)}$ during the matrix factorization update becomes also responsible for generating the data for that block. For the simple TDG in Figure 3 this means that task F_0 generates the data for C_0; and tasks $\mathsf{U}_{0,p}$, $p = 1, 2, 3$, those for C_p.

More importantly, proceeding in this manner we break the synchronization point between each two consecutive levels by allowing level s to commence its execution even before that of level $s-1$ has been completed. This approach can then leverage idle processor cores, performing an speculative execution of tasks pertaining to future levels, which may or may not be necessary depending on the outcome of the stopping test. The test for level s thus becomes an asynchronous signal that can stop the execution of tasks in future levels $s+1, s+2, \ldots$. To favor an early solution of the problem though, we prioritize the executions of the tasks in the lower levels.

To close this discussion, we remark that a solution based on existing task-parallel dense linear algebra libraries, such as `libflame` or PLASMA, is impractical due to 1) the need to exploit the special structure of $A^{(s)}$ in order to reduce the cost of factorization update; and 2) the need of an explicit handling of the solution process to expose additional task-parallelism.

OmpSs implementation of algorithm RLLS. OmpSs is a task-based parallel programming model developed at Barcelona Supercomputing Center [12] that has been successfully applied in the past to the solution of dense linear systems via standard matrix decomposition such as the Cholesky, QR and LU factorizations, on multicore systems; see, e.g., [13].

At execution time, the OmpSs runtime system detects data dependencies between tasks, with the help of the programmer via OpenMP-like compiler directives (pragmas) annotated with clauses that indicate the task operands' directionality (input, output or input/output). OmpSs then generates a task graph, which is leveraged to schedule the tasks to the cores, exploiting the inherent task-level parallelism while fulfilling the graph dependencies.

Our task-parallel solver relies on a sequential implementation of algorithm RLLS and the structure-aware UMF procedure in C, with routines for the kernels/computational tasks offering clean interfaces. From that starting point, we add OmpSs pragmas (`#pragma omp task`) to identify tasks, directionality clauses (`in`, `out` and `inout`) to indicate the character of the routine matrix arguments, and priorities to guide the execution order of the tasks. No attempt to exploit other type of parallelism (e.g., inside the tasks, via calls to a multi-threaded implementation of BLAS) is currently done in our approach.

5 Experimental Results

The goal of this section is to expose the performance benefits of leveraging a task-parallel programming model, such as OmpSs, for the computation of the lazy QR factorization operating with latent matrices. For this reason, our experiments are designed to assess the scalability of the OmpSs factorization code, in a simplified yet practical scenario. In particular, for the evaluation of the task-parallel lazy QR factorization, we generated square latent matrices of orders $n = 14,400$, 20,160, 28,800, 34,560, 40,320, 44,640, and 49,152 for the largest instance. For simplicity, for each problem dimension we set the dimension of all levels to be the same: $m = m_0 = m_1 = m_2 = \ldots$, with values m=600, 840, 1,200, 1,440, 1,680, 1,860, and 2,048

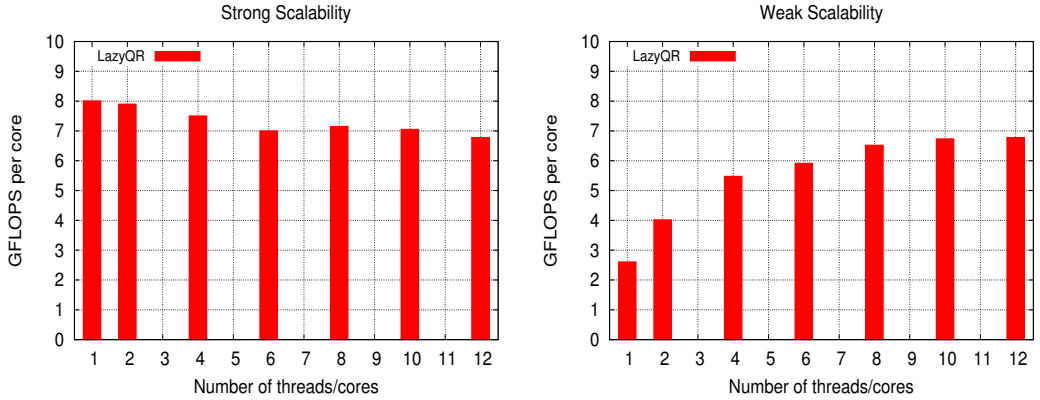

Figure 5: Performance of the task-parallel lazy QR factorization.

growing proportionally with n. This implies that the number of levels s required for convergence is assumed to equal 24 for all problem instances. The memory consumption (footprint) is n^2 plus a minor factor, with n being the largest problem size that is factorized. The evaluation is performed in terms of GFLOPS (billions of flops per second), using the standard cost for the QR factorization.

All the experiments in this section were performed in IEEE double precision arithmetic, on a server equipped with two Intel E5645 hexa-core processors (for a total of 12 cores), at 2.40 GHz, and 48 Gbytes of DDR3 RAM. Our codes were linked with Intel MKL (`composer_xe_2011_sp1`) for the BLAS kernels and OmpSs (version 16.06).

Figure 5 reports the GFLOPS rate attained by the OmpSs version of the lazy QR factorization routines on the Intel 12-core server for two different configurations, corresponding to strong and weak scalability tests. For the evaluation of the strong scalability, we set the problem size to the largest instance, that is n=49,152 (and therefore m=2,048), and then vary the number of threads/cores from 1 to 12. In this scenario, we should observe a decrease in the GFLOPS throughput as the number of cores is increased, reflecting a situation where the problem dimension ($n \times n$) per core is gradually reduced. The left-hand side plot in Figure 5 shows that this is indeed the case. However, the decay is small and mostly occurs for the executions with 4, 6, and 12 threads. Overall the reduction from 1 thread/core and 8 GFLOPS to 12 threads/cores and slightly less than 7 GFLOPS is about 15%, showing a fairly strong scalability for the task-parallel lazy QR factorization. The speed-up with c cores is directly obtained from the results of this experiments by dividing the GFLOPS rates with c cores, multiplied by c, by that obtained with a single core.

For the weak scalability test (in the right-hand side plot in Figure 5), we run the smallest problem instance (n=14,400, m=600) on a single thread/core, and then gradually increase the number of threads/cores while maintaining the problem dimension per core constant. Note that this roughly corresponds to the dimensions that we enumerated in the opening paragraph of this section, with n=14,400, 20,160,...,49,152 respectively executed using 1, 2,..., 12 threads/cores. For this experiment, we observe a rapid increase of the GFLOPS rate for up to 10 threads/cores, but an stagnation for the execution of the largest problem on 12 threads/cores. While this may be a bit disappointing, we believe that a finely-tuned problem-dependent optimization of this case (e.g., by adjusting the blocking parameters used internally in the building blocks of the UMF procedure) could solve this problem.

6 Concluding Remarks and Future Work

We have re-formulated the solution of linear systems with latent data arising in few-body Coulomb problems as a matrix factorization update, which we address via a lazy QR factorization algorithm. Our solution exploits task-parallelism and addresses the reduced degree of parallelism of this problem by proposing an asynchronous execution of multiple linear systems, increasing the concurrency and, therefore, avoiding otherwise idle cores (i.e., wasted computational resources). This approach can thus produce a faster execution, at the expense of performing some speculative calculations that may not be part of the final solution.

As part of future work, we plan to exploit complementary thread-parallelism existing inside the computational building blocks that compose the factorization.

Acknowledgements

The researcher from UJI was supported by project TIN2014-53495-R of the MINECO and FEDER, and project P1-1B2015-26 of UJI. We thank Rocio Carratalá, from UJI, for her help with the evaluation of the building blocks.

References

[1] I. Bray, A. Kheifets, D. V. Fursa, Electrons and photons colliding with atoms: development and application of the convergent close-coupling method, Journal of Physics B: Atomic, Molecular and Optical Physics 35 (2002) R117.

[2] M. S. Mengoue, M. G. K. Njock, B. Piraux, Y. V. Popov, S. A. Zaytsev, Electron-impact Double Ionization of He by Applying the Jacobi Matrix Approach to the Faddeev-Merkuriev equations, Physical Review A 83 (2011) 052708.

[3] A. Frapiccini, J. M. Randazzo, G. Gasaneo, F. D. Colavecchia, A boundary adapted spectral approach for breakup problems, Journal of Physics B: Atomic, Molecular and Optical Physics 43 (10) (2010) 101001.

[4] J. M. Randazzo, A. Frapiccini, L. U. Ancarani, G. Gasaneo, F. D. Colavecchia, Generating Optimal Sturmian Basis Functions for Atomic Problems, Physical Review A 81 (2010) 042520.

[5] M. Baertschy, T. N. Rescigno, W. Isaacs, X. Li, C. McCurdy, Electron-impact ionization of atomic hydrogen, Physical Review A 63 (2) (2001) 022712.

[6] F. D. Colavecchia, Accelerating spectral atomic and molecular collisions methods with graphics processing units, Computer physics communications 185 (7) (2014) 1955–1964.

[7] G. H. Golub, C. F. V. Loan, Matrix Computations, 3rd Edition, The Johns Hopkins University Press, Baltimore, 1996.

[8] B. C. Gunter, R. A. van de Geijn, Parallel out-of-core computation and updating the QR factorization, ACM Trans. Math. Soft. 31 (1) (2005) 60–78.

[9] E. S. Quintana-Ortí, R. A. van de Geijn, Updating an LU factorization with pivoting, ACM Trans. Math. Softw. 35 (2) (2008) 11:1–11:16.

[10] PLASMA project home page, `http://icl.cs.utk.edu/plasma`.

[11] F. G. Van Zee, `libflame`: The Complete Reference, `www.lulu.com`, 2012.

[12] OmpSs project home page, `http://pm.bsc.es/ompss`.

[13] R. M. Badia, J. R. Herrero, J. Labarta, J. M. Pérez, E. S. Quintana-Ortí, G. Quintana-Ortí, Parallelizing dense and banded linear algebra libraries using SMPSs, Concurrency and Computation: Practice and Experience 21 (18) (2009) 2438–2456.