Using Graph Partitioning for Scalable Distributed Quantum Molecular Dynamics

Hristo N. Djidjev[*] Georg Hahn[†] Susan M. Mniszewski[*] Christian F.A. Negre[*]
Anders M.N. Niklasson[*] Vivek B. Sardeshmukh[‡]

Abstract

The simulation of the physical movement of multi-body systems at an atomistic level, with forces calculated from a quantum mechanical description of the electrons, motivates a graph partitioning problem studied in this article. Several advanced algorithms relying on evaluations of matrix polynomials have been published in the literature for such simulations. We aim to use a special type of graph partitioning in order to efficiently parallelize these computations. For this, we create a graph representing the zero-nonzero structure of a thresholded density matrix, and partition that graph into several components. Each separate submatrix (corresponding to each subgraph) is then substituted into the matrix polynomial, and the result for the full matrix polynomial is reassembled at the end from the individual polynomials. This paper starts by introducing a rigorous definition as well as a mathematical justification of this partitioning problem. We assess the performance of several methods to compute graph partitions with respect to both the quality of the partitioning and their runtime.

1 Introduction

The physical movements of multi-body systems on an atomistic level is at the core of molecular dynamics (MD) simulations. Those dynamics take place at the femtosecond (10^{-15} second) time scale and they are incorporated in a larger simulation which typically is of the order of pico- to nanoseconds (10^{-12} to 10^{-9} second). A simple way to conduct MD simulations is to derive all forces from the potential energy surface for all the interacting particles, and to compute molecular trajectories for the multi-particle system by solving Newton's equations numerically. In quantum-based molecular dynamics (QMD) simulations, the electronic structure is based on an underlying quantum mechanical description, from which interatomic forces are calculated.

Several QMD methods are published in the literature for a variety of materials systems. So-called *first principle* methods are capable of simulating a few hundred atoms over a picosecond range. Important approaches of this type include Hartree-Fock or density functional theory. Semiempirical methods such as self-consistent tight-binding techniques increase the applicability to systems of several thousand atoms. In contrast to regular first principle methods, approximate methods are often two to three orders of magnitude faster while still capturing the quantum mechanical behavior (for example, charge transfer, bond formation, excitations, and quantum size effects) of the system.

[*]Los Alamos National Laboratory, Los Alamos, NM 87544, USA
[†]Lancaster University, Bailrigg, Lancaster LA1 4YW, U.K.
[‡]University of Iowa, Computer Science Department, Iowa City, IA 52242, USA

One of the most efficient and widely used methods is density functional based self-consistent tight-binding theory (Elstner et al., 1998; Finnis et al., 1998; Frauenheim et al., 2000). In this approach, the main computational effort stems from the diagonalization of a matrix, the so-called *Hamiltonian matrix*, which encodes the electronic energy of the system. The Hamiltonian matrix is needed in order to construct the *density matrix* describing the electronic structure of the system. Before evaluating the forces at each time step of a QMD simulation, the self-consistent construction of the density marix is carried out. Computing the density matrix requires matrix diagonalization with a computational cost of $O(N^3)$, where N is the dimension of the Hamiltonian. This makes diagonalization only viable for small system sizes. For this reason, the last two decades have seen the development of a number of linear runtime (i.e., $O(N)$) algorithms.

One such linear runtime approach is based on a recursive polynomial expansion of the density matrix (Niklasson, 2002). A linear scaling with the system size for non-metallic systems is achieved by the sparse-matrix second-order spectral projection (SM-SP2) algorithm. On dense or sparse matrices, SM-SP2 competes with or outperforms regular diagonalization schemes with respect to both speed and accuracy (Mniszewski et al., 2015). SM-SP2 uses the expression

$$D = \lim_{i \to \infty} f_i[f_{i-1}[\dots f_0[X_0]\dots]] \tag{1}$$

to compute the density matrix D from the Hamiltonian H, where $f_i(X_i)$ is a quadratic function (either X_i^2 or $2X_i - X_i^2$, depending on $Tr(X_i)$ or $Tr(X_{i+1})$) and the initial matrix X_0 is a linearly modified version of H. Usually, $20 - 30$ iterations suffice to obtain a close approximation of D. Additionally, thresholding is applied to further reduce the computational complexity, where small nonzero elements of the matrix (typically between 10^{-5} to 10^{-7}) are set to zero.

The cost of computing a matrix polynomial P (which is mainly due to the squaring of a matrix) dominates the computational cost of the SM-SP2 algorithm. Since we are interested in performing a large number of time steps (of the order of $10^4 - 10^6$) in a typical QMD simulation, we need to parallelize the evaluation of the matrix polynomials in order to keep the wall-clock time low. However, the significant communication overhead for every iteration causes SM-SP2 to not parallelize well. Linear scaling complexity has been achieved with thresholded blocked sparse matrix algebra (Bock and Challacombe, 2013; Borstnik et al., 2014; Mniszewski et al., 2015; VandeVondele et al., 2012). In our paper, we present an alternative formulation that reduces communication overhead via graph partitioning and enables scalable parallelism. Our basic approach was introduced in Niklasson et al. (2016), but with the main focus being on the physics aspects.

This paper addresses the aforementioned parallel version of SP2 together with an inbuilt partitioning scheme applied to the graph representation of the density matrix, denoted as G-SP2 in the remainder of the article. In particular, the computational aspects of evaluating the matrix polynomial in G-SP2 are investigated. We represent the Hamiltonian (or density) matrix as a graph where atomic orbitals are given as vertices and non-zero interactions become edges, and then partition that graph into parts (or partitions) with the aim to minimize a suitable cost function. The submatrices of the Hamiltonian or density matrix correspond to the divisions of the molecule, and are derived from the graph partitions. We show that applying the full matrix polynomial to the unpartitioned matrix is equivalent to combining the results obtained by applying the matrix polynomials to each submatrix independently.

We show that partitions with overlapping parts (or *halos*) need to be computed in order to obtain accurate results with our graph approach. Naturally, the computational overhead increases with the overlap between partitions. The aim of this work is to minimize the computational cost of the matrix polynomial evaluation through appropriate partitioning schemes. Those schemes will directly minimize the cost of the corresponding polynomial evaluation as opposed to traditional edge cut minimization. In our article, we will experimentally study several algorithms for the aforementioned graph partitioning problem, which we formally introduce first.

We rigorously prove that applying the matrix polynomial to the entire Hamiltonian is equivalent to applying it to the partitioned Hamiltonian matrix and re-assembling the partial solutions, thus justifying our approach to parallelize the computational workload via graph partitioning. Although demonstrated numerically (Niklasson et al., 2016), no such proof exists in the literature to the best of our knowledge.

Regular graph partitioning, defined as the task of separating the vertices of a graph into roughly equal sets that minimize the edge cut between them, has been studied extensively from theoretical and applied aspects. Graph separators form the basis of many divide-and-conquer algorithms for problems such as VLSI design, shortest paths finding, solving sparse systems of linear equations, and approximations of NP-hard problems in theoretical computer science (Lipton and Tarjan, 1979; Aleksandrov and Djidjev, 1996; Miller, 1997; Wulff-Nilsen, 2011). Graph separator algorithms produce balanced partitions for graphs from a given class. The separator is a set of vertices or edges whose removal divides the graph, which usually has size bounded by a small function of the graph parameters. Although theoretical results on graph separators are usually asymptotically optimal, those are still too impractical for applications due to the fact that their leading constants are often too large. A class of graph partitioning tools has been devised which is based on heuristics such as Kernighan-Lin (Kernighan and Lin, 1970) and multilevel optimization Hendrickson and Leland (1995). Available software using those techniques include Chaco (Hendrickson and Leland, 1995), METIS (Karypis and Kumar, 1999), Jostle (Walshaw and Cross, 2000), and KaHIP (Sanders and Schulz, 2011). The advantage of such tools consists in the fact that the input graph is not required to belong to a certain class of graphs (such as planar graphs or graphs of bounded genus). Moreover, those tools usually return partitions of good quality, yet without provable quality bounds. Existing algorithms optimize the size of the cutset as objective function, whereas our work focuses on a new flavor of the graph partitioning problem with overlapping partitions.

Thresholding the matrix elements of the Hamiltonian is essential in order to arrive at a fast way to calculate the density matrix: without thresholding, the resulting matrix would quickly become dense due to fill-in. The accuracy of the collected density matrix will depend on the chosen graph partitioning, though in general the effect is small and the error is controlled mainly by the numerical threshold (Niklasson et al., 2016).

Our approach allows us to avoid communication between processors after each iteration of (1) until the entire polynomial is evaluated. To this end, each processor independently evaluates its assigned polynomial after partitioning and distributing the inital matrix, and the final output is assembled from the computed submatrices. In this article, different algorithmic approaches for computing graph partitions are assessed with respect to the aforementioned objective function and their computational effort. Importantly, we analyze the tradeoff between the additional computational costs for computing graph partitions before running the SP2 algorithm in parallel, and carrying out regular molecular dynamics. We also investigate the optimal number of partitions as a function of the graph size.

The structure of this paper is as follows. Section 2 introduces the mathematical foundations for partitioning the evaluation of matrix polynomials, states our algorithm including a proof of correctness, and defines the graph partitioning problem we consider. Algorithms for constructing such partitions and their implementations are discussed in Section 3. Experimental results for several physical test systems are given in Section 4. Section 5 discusses our results. Proofs for Section 2 can be found in Appendix A, and Appendix B presents further experimental results.

A preliminary version of this article has been published as a conference paper in the *SIAM Workshop on Combinatorial Scientific Computing (CSC16)*, see Djidjev et al. (2016). This article is an extension of the work of Djidjev et al. (2016), and includes proofs of all theoretical results of Section 2 in Appendix A, pseudo-code of our simulated annealing approach in Section 3.2, a visualization of the relationship between the graph structure of a molecule and its partitioned graph representation in Section 4.3, and more detailed performance data used for all experiments in Appendix B.

2 Evaluating Matrix Polynomials on Partitions

We define a thresholded matrix polynomial, justify its parallelized evaluation, present an algorithm to evaluate a matrix polynomial in a parallelized fashion, and conclude by defining the cost function for an implied graph partitioning problem.

We encode the zero-nonzero structure of a symmetric matrix $X = \{x_{ij}\}$ as a graph $G(X)$, called the *sparsity graph* of X. $G(X)$ contains a vertex for each row (or column) in X, and $G(X)$ contains an edge between vertices i and j if and only if $x_{ij} \neq 0$. We now generalize the matrix polynomial defined in (1) for any symmetric $n \times n$ matrix A. Denote the superposition of operators of the type

$$P = P_1 \circ T_1 \circ \ldots \circ P_s \circ T_s \tag{2}$$

as a *thresholded matrix polynomial* of degree $m = 2^s$, where T_i is a thresholding operation and P_i is a polynomial of degree 2. For any graph I, we formally define T_i as the graph operator (with associated edge set $E(T_i)$) such that $T_i(I)$ is a graph with a vertex set $V(I)$ and an edge set $E(I) \setminus T_i$.

The application of a superpositioned operator P of the type (2) to a matrix A of appropriate dimension is denoted as $P(A)$. Analogously to (2), P is composed of polynomials P_i and thresholding operations T_i. For our SM-SP2 application, the Hamiltonian is A and the density matrix is $P(A)$.

For any matrix A, we define all matrices B which have the same zero-nonzero structure as A (that is, $G(A) = G(B)$) to be in the structure class $\mathcal{M}(A)$.

Let $G = G(A)$ for a matrix A and let P be a thresholded matrix polynomial. With $\mathcal{P}(G)$ we denote the minimal graph with the same vertices as G such that if $P(B)|_{vw} \neq 0$ for any matrix $B \in \mathcal{M}(A)$ and any v, w, then there is an edge $(v, w) \in E(P(G))$. We interpret $\mathcal{P}(G)$ as the worst-case zero-nonzero structure of $P(A)$ which excludes cancellations resulting from the addition of opposite-sign numbers, thus resulting in coincidental zeros. All diagonal elements of A are assumed to be non-zero, and $E(T_i)$ is assumed to not contain a loop edge.

Assume we are given a collection of partitions $\Pi = \{\Pi_1, \ldots, \Pi_q\}$. Each partition $\Pi_i = U_i \cup W_i$ is a union of a *core* vertex set U_i and a *halo* vertex set W_i. Given the two following conditions hold true, we call Π a *CH-partition* (or core-halo partition):

1. $\bigcup_i U_i = V(G),\ U_i \cap U_j = \emptyset$ for all $i \neq j$;

2. neighbors of vertices in U_i that are themselves not in U_i are contained in W_i.

Let H_{U_i} be the subgraph of $H = \mathcal{P}(G)$ induced by all neighbors of U_i in H. We combine all rows and columns of A that correspond to vertices of $V(H_{U_i})$ in a submatrix A_{U_i} of A. The following lemma shows that $P(A)$ can be computed on submatrices of the Hamiltonian.

Lemma 1. *For any $v \in U_i$ and any neighbor w of v in $\mathcal{P}(G)$, the element of $P(A)$ corresponding to the edge (v, w) of $\mathcal{P}(G)$ is equal to the element of $P(A_{U_i})$ corresponding to the edge (v, w) of H_i.*

Lemma 1 justifies the parallelized evaluation of a matrix polynomial.

Let us apply the aforementioned results to a Hamiltonian matrix A in a QMD simulation. In this case, we can assume that the sparsity structure of the density matrix D from the previous QMD simulation step is being passed on to $P(A)$. We can thus approximate $H = \mathcal{P}(G)$ with $G(D)$ (the graph H is unknown until $P(A)$ is computed). The halos can also be taken from H in practice.

We propose the following algorithm for computing $P(A)$ from $H = G(D)$:

1. Divide $V(G)$ into q disjoint sets $\{U_1, \ldots, U_q\}$ and define a CH-partition $\Pi = \{\Pi_1, \ldots, \Pi_q\}$, where Π_i has core U_i and halo $N(U_i, H) \setminus U_i$;

2. Construct submatrices A_{U_i} for all $i = 1, \ldots, q$;

3. Compute $\mathcal{P}(A_{U_i})$ for all i independently using dense matrix algebra;

4. Define $\mathcal{P}(A)$ as a matrix whose i-th row has nonzero elements equal to the corresponding elements of the j-th row of $\mathcal{P}(A_{U_k})$, where U_k is the set containing vertex i and j is the row in A_{U_k} corresponding to the i-th row in A.

This algorithm computes $P(A)$ as demonstrated in Lemma 1.

The computational bottleneck of the algorithm for $P(A)$ is caused by the dense matrix-matrix multiplication required to compute $\mathcal{P}(A_{U_i})$ for all i in step (iii).

To be precise, according to (2), computing $\mathcal{P}(A_{U_i})$ takes $s(c_i + h_i)^3$ operations, where c_i and h_i are the size of the core and the halo of Π_i and s is the number of superpositioned operators (see (2)). In this calculation, the computational effort for thresholding some matrix elements is excluded, since this effort is quadratic in the worst case and linear in $c_i + h_i$ in average cases. We observe that a CH-partition which minimizes the effort to compute $P(A)$ also minimizes $\sum_{i=1}^{q}(c_i + h_i)^3$ due to the fact that s is independent of Π.

This observation motivates our *CH-partitioning problem*, defined as follows: For an undirected graph G and an integer $q \geq 2$, split G into q parts $\Pi_1, \ldots, \Pi_q$ such that

$$\sum_{i=1}^{q}(c_i + h_i)^3 \tag{3}$$

is minimized, where Π_i has a core U_i of size c_i and a halo $N(U_i, G) \setminus U_i$ of size h_i.

As an example, the optimal CH-partitioning for a star graph of n vertices has a single non-empty part containing all vertices: this part is composed of the central vertex, whose halo contains all other vertices. In contrast, a standard (edge cut) partitioning will have n parts, thus demonstrating that a CH-partitioning minimizing (3) can be quite different from a standard balanced partitioning.

3 Algorithms for Graph Partitioning Considered in our Study

In this section we investigate the ability of existing graph partitioning packages, as well as our own heuristic algorithm, for computing CH-partitions that minimize the objective function (3). Those algorithms are: *METIS* and *hMETIS* due to their widespread use, and *KaHIP* based on its convincing performance at the 10th DIMACS Implementation Challenge Bader et al. (2013).

3.1 Edge Cut Graph Partitioning

We observe that we obtain $|V(G)| + \sum_i h_i$ when leaving out the cubes in the objective function (3), thus making it necessary to minimize the sum of the halo nodes over all parts.

Regular graph partitions and CH-partitions are related. Suppose we are given a regular partition P. For any part in P, we can define a core corresponding to that part, and a halo consisting of all adjacent vertices of the core vertices (excluding the core vertices themselves). We define the CH-partition Π to consist of precisely those parts and halos for any element of P. It must then be true that either v or w is a halo vertex for any cut edge (v, w) of P. Conversely, there exists a core vertex w such that (v, w) is a cut edge for any halo vertex v belonging to some part in Π.

This shows that the cut edges of P and the set of halo nodes in Π are related but not equal. We observe that another measure, the *total communication volume*, exactly corresponds to the sum of halo nodes. Certain tools like *METIS* allow us to optimize with respect to the total communication volume. By ignoring the cubes in (3), we aim to study how well CH-partitions can be produced by regular graph partitioning tools. Additionally, we improve the solutions obtained by standard graph partitioning tools with our own heuristic in Section 3.2. The three following algorithms will be used:

3.1.1 METIS

METIS (Karypis and Kumar, 1999) uses a three-phase multilevel approach to perform graph partitioning:

1. Starting from the original graph $G = G_0$, *METIS* generates a graph sequence $G_0, G_1, \ldots, G_n$ to coarsen the input graph. The coarsening ends with a suitably small graph G_n (typically less than 100 vertices).
2. An algorithm of choice is used to partition G_n.
3. Using the sequence $G_{n-1}, \ldots, G_1$, the partitions are projected back from G_n to G_0.

Additionally, *METIS* employs a refinement algorithm such as the one of Fiduccia-Mattheyses (Fiduccia and Mattheyses, 1982) to improve the partitioning after each projection. This is necessary since the finer the partition, the more degrees of freedom it has during the uncoarsening phase. Several tuning parameters can be set in *METIS*, including the size of G_n, the coarsening algorithm, and the algorithm used for partitioning G_n.

3.1.2 KaHIP

Several multilevel graph partitioning algorithms are combined in *KaHIP* Sanders and Schulz (2013). *KaHIP* works similarly to *METIS*. A given input graph is first contracted, partitions are computed on the highest contraction level, and the partitions obtained in this way are projected back to coarser levels, where refinement algorithms are used to enhance the solutions. *KaHIP* offers max-flow/min-cut (Sanders and Schulz, 2011; Ford Jr. and Fulkerson, 1956), Fiduccia-Mattheyses (Fiduccia and Mattheyses, 1982) or F-cycles (Sanders and Schulz, 2011) for local improvement of the solution.

3.1.3 Hypergraph partitioning

A hypergraph formulation is an alternative approach to the classical interpretation of the density matrix as an adjacency matrix (where each nonzero interaction in the density matrix is an undirected and unweighted edge).

The set of all neighbors of a vertex form a single hyperedge in the hypergraph formulation. The main advantage of using hyperedges consists in the fact that minimizing the edge-cut with respect to hyperedges results in either all or zero vertices being included in a partition. This automatically ensures that halos are included in a partition together with the core vertex.

We use *hMETIS* of Karypis and Kumar (2000) in order to compute hypergraph partitionings. *hMETIS* is the hypergraph analog of *METIS*.

3.2 Refinement with Simulated Annealing

The objective function (3) we aim to minimize differs from the size of the edge or hyperedge cut minimized by standard graph and hypergraph partitioning algorithms. With the help the algorithm derived in this section we explicitly minimize (3).

A standard tool in optimization is the probabilistic algorithm of (Kirkpatrick et al., 1983), called simulated annealing (*SA*). SA iteratively proposes random modifications to an existing solution in order to improve it, and thus optimizes without gradients. If a proposed modification (or move) does not immediately lower the objective function, it might still be accepted with a certain *acceptance probability*. The acceptance probability is proportional to the magnitude of the (unfavorable) increase in the objective function, and antiproportional to the runtime. The latter makes it more likely for SA to accept unfavorable moves in the exploration phase at the start of each run, and it is implemented using a strictly decreasing *temperature* function. Modifications to the existing solution which further minimize the objective function are always accepted.

A fixed number of iterations, a vanishingly small temperature, or the lack of further improvements in the solution over a certain number of iterations can be employed as stopping criteria for SA.

We test the following proposal functions that return modifications to existing solutions, where a partition P is simply a set of nodes:

1. Select a random partition P, select one of its halo nodes v at random and move v into partition P.

2. Select a random partition P, select one of its nodes v at random and move v into P.

3. Like (2.) but select a random halo node v of partition P.

4. Select the partition P with most halo nodes and (a) move a random node v into P, (b) make a random halo node of P a core node, or (c) move any node of P to another partition.

5. Like (4.) using the partition P with the largest sum of core and halo nodes.

Many more sensible proposal functions could be devised. However, in our experiments we observed that the above proposals result in a similar behavior of SA, with the best tradeoff between speed and performance being achieved by scheme (3.).

Algorithm 1 states the SA implementation we use in our experiments. Any regular (edge cut) partitioning (e.g., obtained with *METIS*), and even a random partitioning, serves as input Π (the set of partitions) to Algorithm 1. The SA algorithm runs over a fixed number of N iterations. In each iteration, we randomly select a partition $\pi \in \Pi$ as well as a random edge joining a core vertex v with a halo vertex w in π. Afterwards, π is updated with w being a core vertex. After storing the new partitioning in a set Π', both Π and Π' are assessed. For this we compute the change Δ in the objective function (3) between Π' and Π, which is used to update the acceptance probability p. We accept Π' (thus overwriting $\Pi := \Pi'$) with probability p. Note that $p > 1$ if $\Delta < 0$, thus leading to a guaranteed acceptance of a proposal that directly improves (3). Afterwards, the iteration is repeated.

SA is run with a maximal number of iterations $N = 100$ as stopping criterion, and the temperature function is chosen as $t(i) = 1/i$. In practice, we first threshold a density matrix (see Section 2),

Algorithm 1: Simulated Annealing

```
Input: Graph G, number of iterations $N$, initial partitioning $\Pi$
Output: Updated partitioning $\Pi$
1 Select a temperature function $t(i) = 1/i$ ;
2 for $i = 1$ to $N$ do
3     Select random partition $\pi$ in $\Pi$ and a random core-halo edge $(v, w)$ in $\pi$;
4     Make $w$ a core vertex of $\pi$ and update the halo of $\pi$;
5     Compute the values $S$ and $S'$ of (3) for $\Pi$ and $\Pi'$, respectively, and set $\Delta = S' - S$;
6     Compute $p = \exp(-\Delta/t(i))$;
7     Set $\Pi = \Pi'$ with probability $\min(1, p)$;
8 end
9 Output partitioning $\Pi$;
```

convert it to a graph, compute regular edge cut partitions for the converted graph with a standard software, and post-process the partitions with SA.

4 Experiments

Using three measures the quality of the CH-partitions returned by the algorithms of Section 3 is evaluated. Those measures are the objective function (3), the algorithmic runtime, and the number of MPI ranks and threads in an assessment of the scaling behavior of the G-SP2 algorithm (for one fixed system). We employ graphs derived from representations of actual molecules as opposed to simulated random graphs.

4.1 Parameter Choices for METIS and hMETIS

Using a grid search over sensible values, we tune the parameters of *METIS* and *hMETIS*, and will keep the set of parameters yielding the best average performance for all systems considered in this section fixed throughout the remainder of the simulations.

The default multilevel k-way partitioning as well as the default sorted heavy-edge matching for coarsening the graph were employed to run *METIS*. Importantly, the user can choose to minimize either the *edge cut* or the *total communication volume* of the partitioning within the k-way partitioning routine of *METIS*. As our definition of the *sum of halo nodes* in Section 2 is equivalent to the definition of the total communication volume of *METIS*, we choose this option.

We employ *hMETIS* with the followed parameters: we use recursive bisectioning (instead of k-way partitioning) with vertex grouping scheme *Ctype*$= 1$ (meaning the hybrid first-choice scheme HFC), Fiduccia-Mattheyses refinement (refinement heuristic parameter set to 1), and V-cycle refinement on each intermediate solution (V-cycle refinement parameter set to 3). These parameters are explained in the *hMetis* manual Karypis and Kumar (1998).

4.2 A Collection of Test Graphs Derived from Molecular Systems

This section uses a selection of physical test systems to evaluate all algorithms of Section 3, which were chosen to represent a variety of realistic scenarios where graph partitioning can be applied to MD simulations. We demonstrate how the graph structure influences the results (Section 4.3) and additionally provide insights into the physics of each test system.

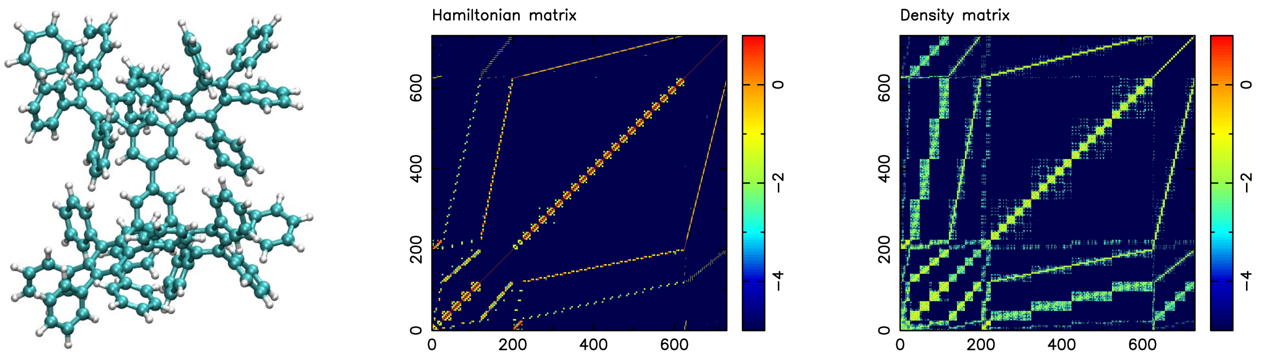


Figure 1: Molecular representation of phenyl dendrimer (left) using cyan and white spheres for carbon and hydrogen atoms, respectively. Hamiltonian matrix as 2D representation (middle) and thresholded density matrix (right). All plots show $\log_{10}$ of the absolute values of all matrix elements. The SM-SP2 algorithm was used to compute the density matrix. Figure taken from Djidjev et al. (2016). Copyright ©2016 Society for Industrial and Applied Mathematics. Reprinted with permission. All rights reserved.

Table 1: Physical systems of our study: number of vertices n in the graph and number of edges m. Table taken from Djidjev et al. (2016). Copyright ©2016 Society for Industrial and Applied Mathematics. Reprinted with permission. All rights reserved.

Name	n	m	m/n	Description
polyethylene dense crystal	18432	4112189	223.1	crystal molecule in water solvent (low threshold)
polyethylene sparse crystal	18432	812343	44.1	crystal molecule in water solvent (high threshold)
phenyl dendrimer	730	31147	42.7	polyphenylene branched molecule
polyalanine 189	31941	1879751	58.9	poly-alanine protein solvated in water
peptide 1aft	385	1833	4.76	ribonucleoside-diphosphate reductase protein
polyethylene chain 1024	12288	290816	23.7	chain of polymer molecule, almost 1-dimensional
polyalanine 289	41185	1827256	44.4	large protein in water solvent
peptide trp cage	16863	176300	10.5	smallest protein with ability to fold (in water)
urea crystal	3584	109067	30.4	organic compound in living organisms

A dendrimer molecule with 22 covalently bonded phenyl groups of solely C and H atoms is schematically shown in Figure 1 (left). The graph of the dendrimer molecule has 730 vertices, composed of 262 atoms and 730 orbitals.

Figure 1 (middle) displays the absolute values of the Hamiltonian matrix for the dendrimer system. Figure 1 (right) displays the density matrix encoding the physical properties of the system, which is obtained by applying the SM-SP2 algorithm to the Hamiltonian.

We threshold the density matrix at 10^{-5} to convert it into a graph needed to find meaningful physical components via graph partitioning. This is done with all systems of Table 1 to arrive at their adjacency matrices. The first column of Table 1 displays the molecule name, the number of vertices n and edges m (second and third column) of its graph representation, and its average vertex degree m/n (fourth column). A short description of the molecule can be found in the last column.

We consider four topologically different types of molecular systems (see Figure 2) as test beds for the partitioning algorithms. This is to provide a wider range of test systems for our algorithms.

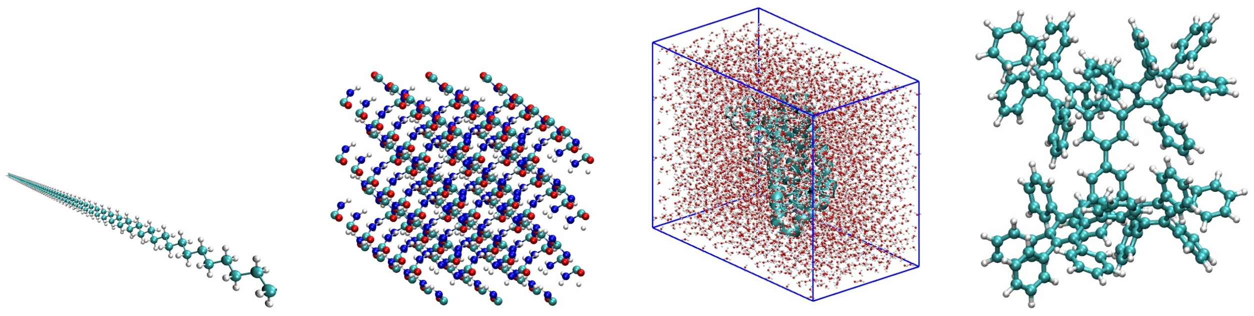


Figure 2: Molecular systems of this study: polyethyene linear chain (first plot), urea crystal (second plot), 189 residue polyalanine solvated in a water box (third plot), and phenyl dendrimer molecule (fourth plot). Cyan, blue, red and white spheres represent carbon, nitrogen, oxygen and hydrogen atoms, respectively. Figure taken from Djidjev et al. (2016). Copyright ©2016 Society for Industrial and Applied Mathematics. Reprinted with permission. All rights reserved.

Figure 2 displays a one dimensional system in the first panel (polyethyene linear chain with repeated CH_2 units), an anisotropic pristine 3D urea crystal (second panel), a polyalanine molecule solvated in water (third panel) with a typical α-helix secondary structure, and a dendrimeric system with a fractal arrangement of phenyl rings (fourth panel) to challenge our partitioning algorithms.

4.3 Comparison of the Partitioning Algorithms

Six methods are tested to partition each graph of Table 1 into 16 parts:

1. *METIS* with parameters of Section 4.1;
2. *METIS* with subsequent simulated annealing (SA);
3. *hMETIS*;
4. *hMETIS* with subsequent SA;
5. *KaHIP*;
6. *KaHIP* with subsequent SA.

As before, the sum of cubes (3) criterion is used to assess the effectiveness of each method.

Figures 3 and 4 show experimental results. We observe that all algorithms perform well (with the exception of the first two systems), and that *METIS* and *KaHIP* are significantly faster than *hMETIS*. Importantly, post-processing with SA seems to improve solutions in almost all cases at negligible additional runtime, and is thus recommended.

Surprisingly, CH-partitions seem to pose a challenge for *hMETIS* as its solutions are usually worse than those of the other two methods, and its runtime significantly exceeds the one of the other methods (an explanation of this remains for further reseach). We conclude that for these two reasons, *hMETIS* seems unsuited for QMD simulations over longer time intervals, which is the aim of this work.

Although solutions returned by *KaHIP* are of very good quality (measured with the sum of cubes criterion), the combination of *METIS* and SA still outperforms *KaHIP* while also having a shorter combined runtime.

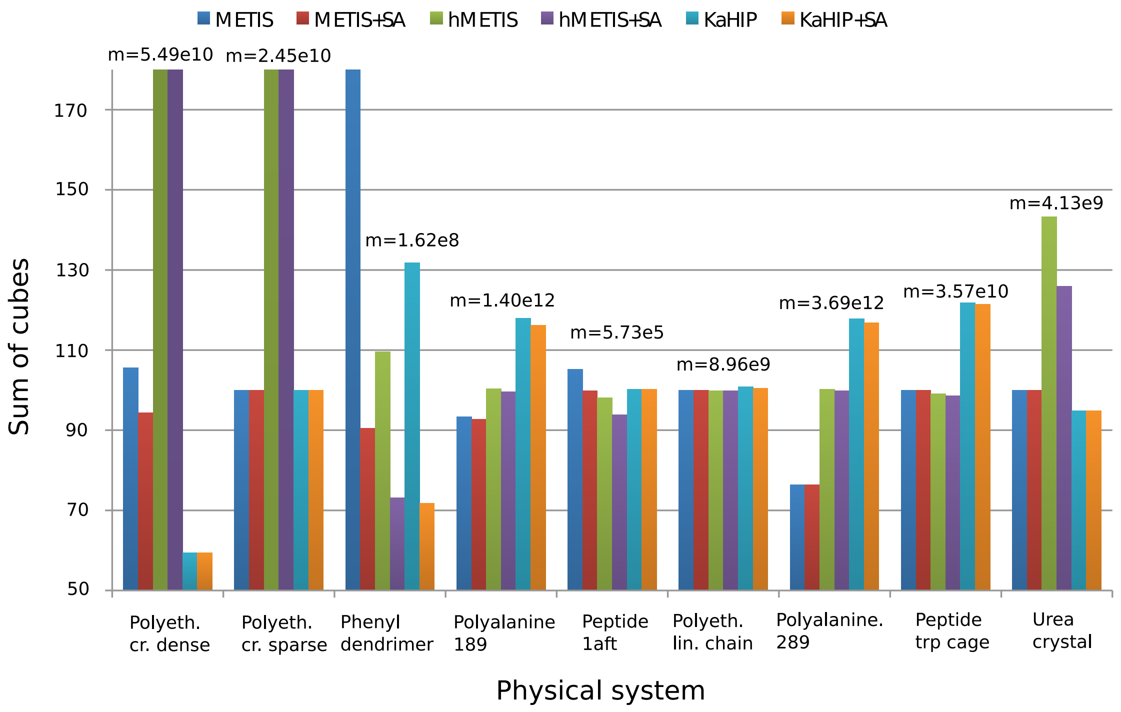


Figure 3: Sum of cubes performance measure to evaluate partitions. Values are normalized to have a median of 100. To make the chart more informative, very large values are truncated, though all exact values can be found in (Djidjev et al., 2016, Table 2). Figure taken from Djidjev et al. (2016). Copyright ©2016 Society for Industrial and Applied Mathematics. Reprinted with permission. All rights reserved.

The behavior of our algorithms seems to be dependent on the sparsity of the graph of a physical system. First, *METIS* is able to outperform *hMETIS* for denser graphs, but not for sparser ones. Second, the post-improvement of partitions with SA seems to be especially effective for dense graphs, which can be explained with the fact that dense graphs offer more possibilities to reassign and optimize edges than sparse graphs. This can be seen when applying the combination of *METIS+SA* to the dense dendrimer system, see (Djidjev et al., 2016, Table 2) for details.

Figure 5 visualizes the relationship between the graph structure of a molecule (for the phenyl dendrimer molecule of Table 1) and its graph partitioning obtained through *METIS* and SA. After partitioning the molecular graph structure, the fractal-like structure of the phenyl dendrimer molecule and its dense components become clearly visible, as well as its sparse connections to other dense components. This structure is what our algorithm exploits to reduce the computations of the density matrix. Interestingly, SA sometimes dissolves entire partitions (meaning it produces partitions with no vertices), since such imbalanced partitions still yield a further decrease of the objective function (3).

4.4 Parallelized Implementation of G-SP2

We also assess the quality of the CH-partitions by measuring the speed-up when parallelizing the G-SP2 algorithm and applying it to real physical systems. In order to do this, the implementation of the G-SP2 algorithm of (Niklasson et al., 2016) was modified to incorporate the graph partitioning step.

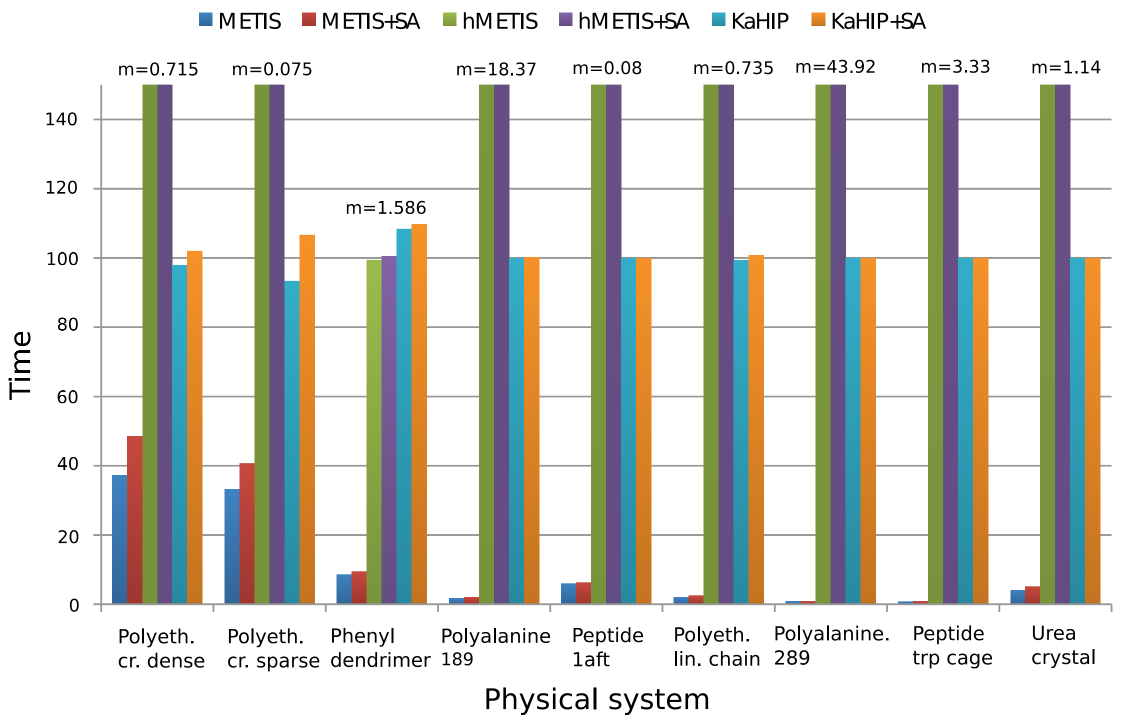


Figure 4: Computing time for partitioning. Test systems of Table 1. We use the formatting of Figure 3 to handle the big discrepancy between values for different graphs. Figure taken from Djidjev et al. (2016). Copyright ©2016 Society for Industrial and Applied Mathematics. Reprinted with permission. All rights reserved.

In particular, to measure the speed-up, we run *METIS* and *METIS+SA* on the *polyalanine 259* protein system of Table 1 and record the obtained CH-partitions. Computations were carried out with the Wolf IC cluster of Los Alamos National Laboratory, whose computing nodes have 2 sockets each containing an 8-core Intel Xeon SandyBridge E5-2670, amounting to a total of 16 cores per computing node. A total of 32 GB of RAM are shared between the sixteen cores on each node, which are connected using a Qlogic Infiniband (IB) Quad Data Rate (QDR) network in a fat tree topology using a 7300 Series switch. Parallelization across nodes was done with OpenMPI, and parallelization across cores within a node was done with OpenMP.

The sum of cubes measure and the computing time are displayed in Figure 6 as a function of the number of CH-partitions for the *polyalanine 259* protein system. We observe in Figure 6 (top) that the total effort of G-SP2, measured with the sum of cubes criterion, decreases monotonically as a function of the number of partitions and parallelized subproblems.

Figure 6 (bottom) displays the computing time for the graph partitioning step alone. We observe that the partitioning effort increases with the number of partitions. The steps occurring at 65, 129, 257 etc. partitions in the plot can easily be explained: Every time the number of partitions surpasses a power of two, the multilevel algorithm which *METIS* is based on bisects the partitioning problem into one more (recursive) layer. Figure 6 (bottom) also displays that employing the SA post-processing step only adds a minimal additional effort to the overall computation (when compared to the graph partitioning step alone). The usage of SA thus seems very sensible in light of the improvements it achieves when applied to the edge cut optimized partitions computed by a conventional algorithm. To summarize, Figure 6 demonstrates that the total effort of the G-SP2

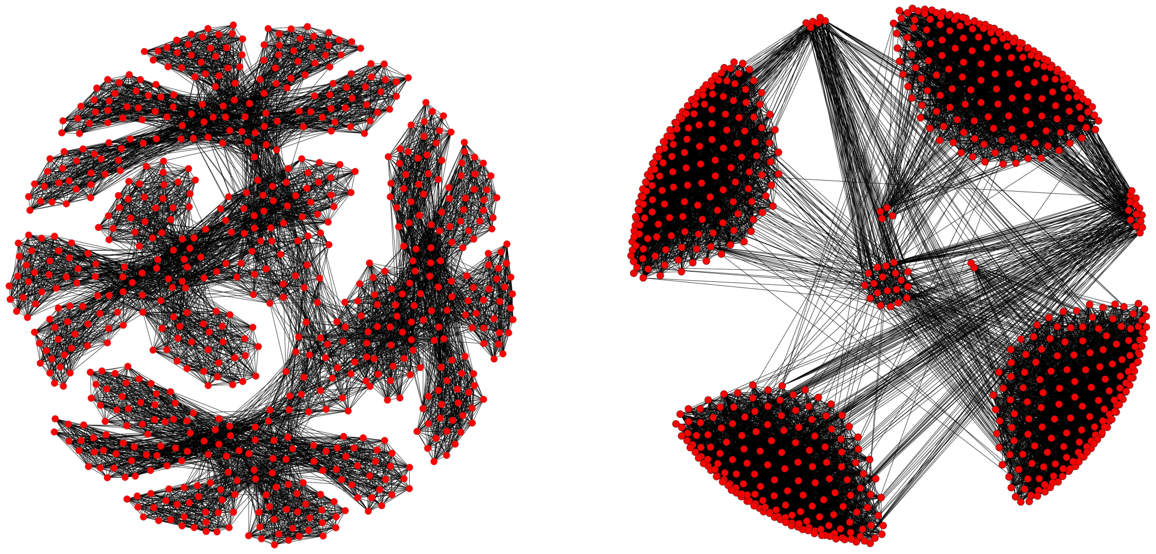


Figure 5: Left: Original graph extracted from the density matrix for the phenyl dendrimer molecular structure. Note the fractal-like structure of the graph. Right: Rearranged graph by the partitions resulting from the METIS + SA algorithms. Only edges with weights larger than 0.01 were kept to ease visualization.

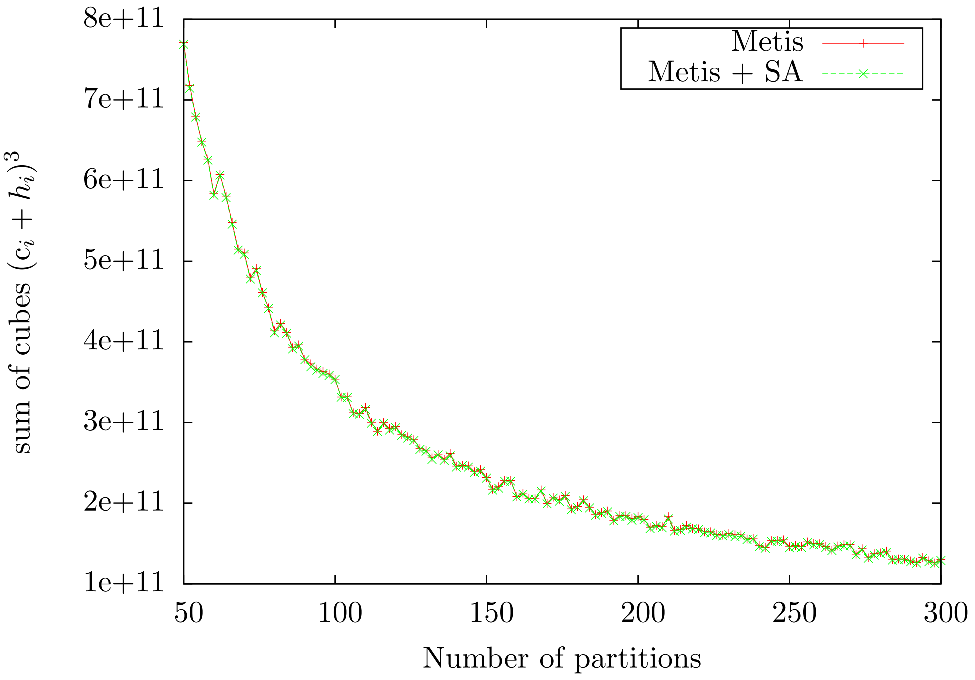
algorithm decreases when applied in parallel despite the increasing effort to compute a partitioning.

4.5 Single Node SM-SP2 versus Parallelized Implementation of G-SP2

Figure 7 aims to quantify the computational savings over single node G-SP2 when running our proposed parallel SP2 algorithm. For this, Figure 7 compares the runtime for our parallel G-SP2 on $1-32$ nodes against a threaded single node implementation of SM-SP2. This is due to the fact that the communication overhead exceeds the gain obtained by the extra computing power in a multi-node implementation, which is also the main motivation for developing G-SP2. As before, we employed *METIS* with parameters specified in Section 4.1 together with SA for post-processing. The test system is again the polyalanine 259 molecule.

Figure 7 shows that, as expected, both an increasing number of nodes and an increasing number of partitions decreases the G-SP2 runtime. When only few nodes are used for parallelization, the decrease in runtime is most pronounced since then, increasing the number of parallel nodes causes the runtime to drop sharply. For a higher number of nodes the curves somewhat flatten out.

Due to the overhead from the parallel G-SP2 computation, the parallelized run of SP2 is actually slower than the SM-SP2 computation on a single node for low numbers of nodes (between 4 and 16 nodes depending on the number of partitions). The runtime decreases with an increase in the number of nodes. Eventually, the effort falls below the one of a single node implementation. For the particular physical system of the polyalanine 259 molecule, a computational speed-up is only observed for at least 4 nodes.



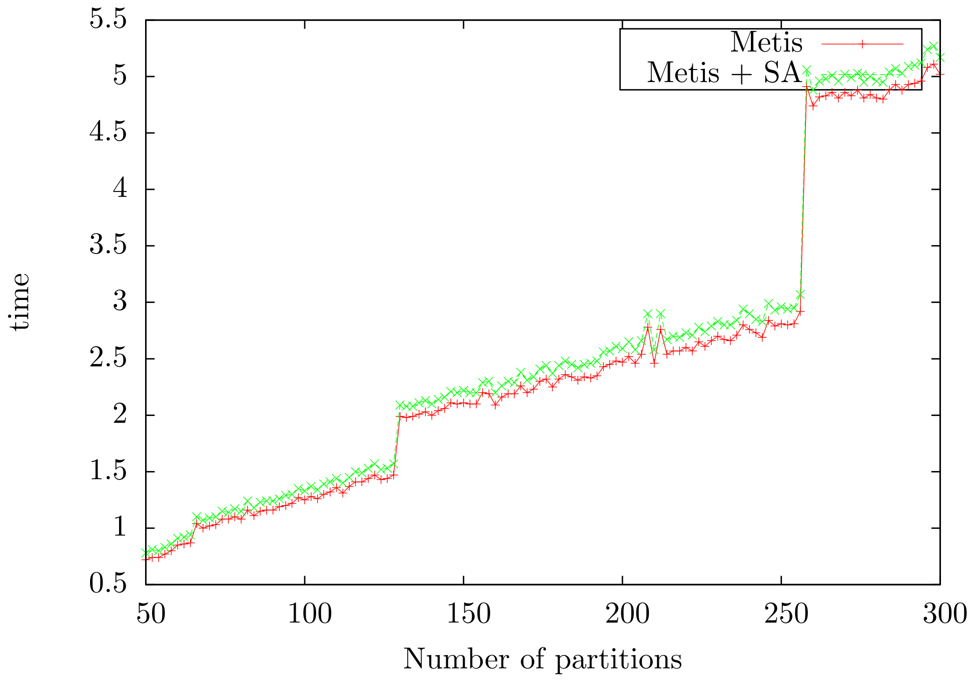


Figure 6: Sum of cubes measure (top) and runtime to find the partitions (bottom) as a function of the number of partitions. Figure taken from Djidjev et al. (2016). Copyright ©2016 Society for Industrial and Applied Mathematics. Reprinted with permission. All rights reserved.

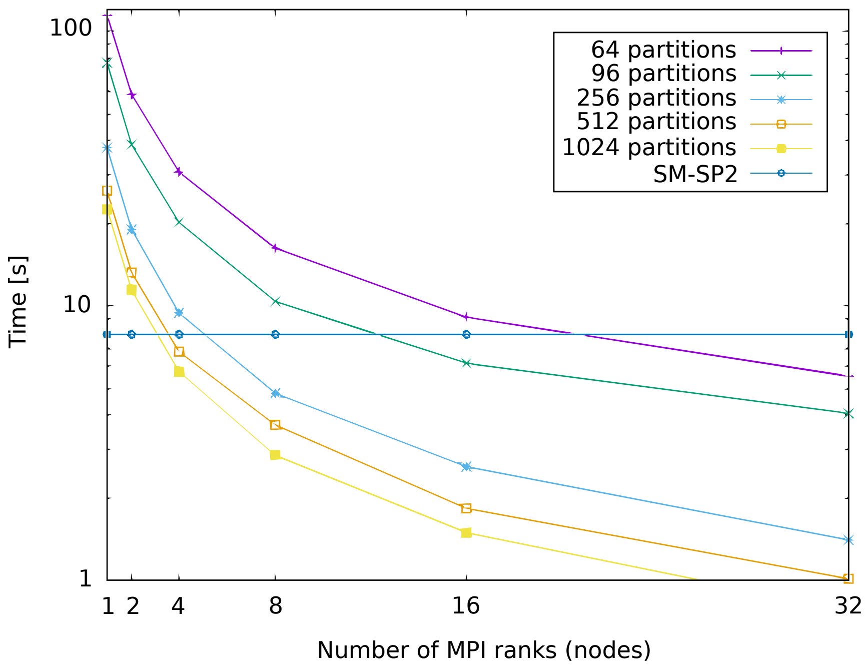


Figure 7: Runtime of parallelized G-SP2 as a function of the number of nodes. Different numbers of partitions. G-SP2 applied to the polyalanine 259 molecule. Figure taken from Djidjev et al. (2016). Copyright ©2016 Society for Industrial and Applied Mathematics. Reprinted with permission. All rights reserved.

4.6 Relationship between Molecular System and Partitions

Figure 8 does not seem to exhibit a correlation between the molecular connectivity (measured with the average graph degree) and the normalized number of operations (NNO), defined as the sum of cubes criterion (3) normalized by the complexity n^3 of the dense matrix-matrix multiplication. The observation also applies to the polyethylene dense crystal, whose normalized number of operations remains low although its average degree is high. Similar observations can be made for other molecules with a smaller average degree.

Our algorithm is capable of finding the lowest NNO for 1-dimensional systems such as the polyethylene linear chain and the polyethylene sparse crystal. According to Bunn (1939), regular agglomerates of polyethylene chains align along a particular direction with a large chain-to-chain distance. We conjecture that the reason for this lies in the sparsity of the system.

We do not observe any advantage of our approach (measured via NNO) for *regular* systems (polyethylene linear chain, polyethylene sparse crystal, polyethylene dense crystal and urea crystal).

A difficult case for our graph partitioning task is the phenyl dendrimer (expressed through its high NNO values). This is due to the fractal-like structure of the graph associated with the molecule. According to (Djidjev et al., 2016, Table 2), *METIS* and SA together seem to yield the best runtime for this molecule.

Another class with a large NNO are proteins (solvated polyalanines). We conjecture that the large average node degree (in comparison to peptides, i.e. small proteins) is responsible for the large NNO measurements. Based on Figure 8, we further conjecture that the difference of maximum and minimum partition norms $(\max - \min)/n$ (MMPN) is correlated with the NNO. Here, the number of vertices is denoted as n, and the maximal and minimal sizes of the obtained partitions are

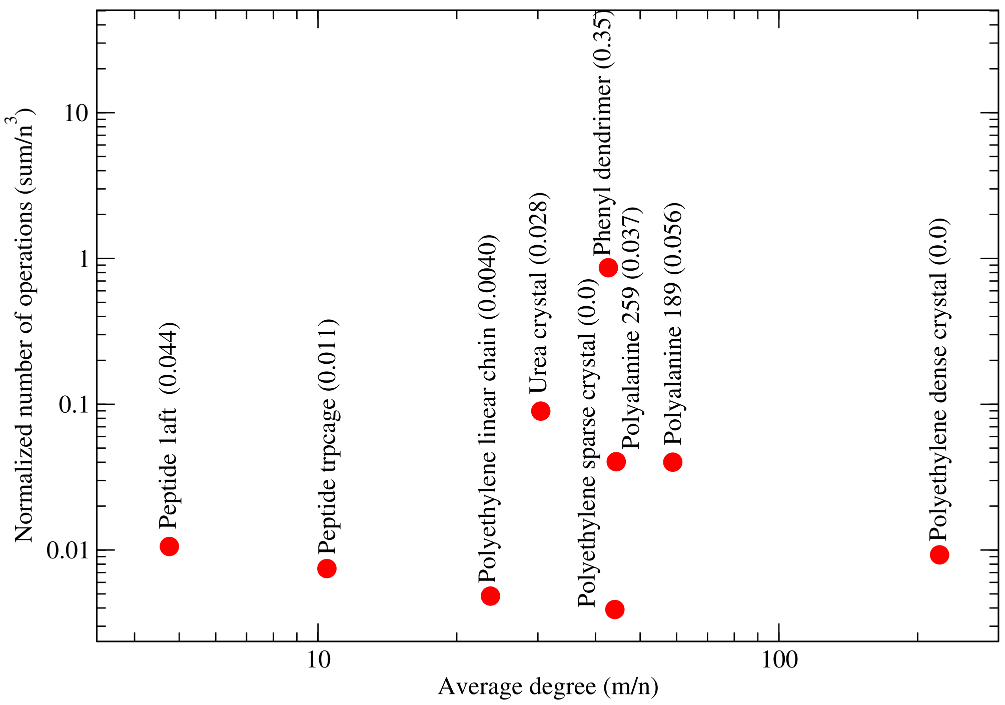


Figure 8: Sum of cubes criterion (3) normalized by the complexity n^3 of the dense matrix-matrix multiplication. Number of vertices n, number of edges m, and average degree m/n. Test systems of Table 1. Brackets show fractions of $(\max - \min)/n$ (MMPN). For the combination *METIS* and SA similar trends are expected. Figure taken from Djidjev et al. (2016). Copyright ©2016 Society for Industrial and Applied Mathematics. Reprinted with permission. All rights reserved.

denoted as max and min, respectively. The conjectured correlation is clearly visible in Figure 8: the dendrimer tends to both large MMPN and NNO values, proteins exhibit intermediate values of both MMPN and NNO and finally, we observe low values of both MMPN and NNO for sparse ordered systems such as polyethyene chains.

5 Discussion

This paper speeds up the computation of the density matrix in MD simulations through parallelization, informed by graph partitioning applied to the structure graph underlying a molecule. Our experimental results are based on graphs derived from density matrices of physical systems.

In our article we focus on a certain flavor of the classical graph partitioning problem arising from molecular dynamics simulations. In contrast to classical edge cut partitioning, we minimize partitions with respect to both the number of their core vertices and the number of their neighbors in adjacent partitions (halos). To the best of our knowledge, this type of graph partitioning (which we coin *CH-(core-halo)-partitioning*) has not been studied previously.

This work makes two contributions. First, the CH-partitioning problem under consideration is mathematically described and justified. We prove that the partitioned evaluation of a matrix polynomial is equivalent to the evaluation of the original (unpartitioned) matrix given a sufficient condition is satisfied. Second, we evaluate several approaches for computing CH-partitions using three error criteria: the total computational effort, the maximal effort per processor, and the overall computational runtime. Special focus is given to the post-processing of partitions obtained with conventional graph partitioning algorithms for which we use our own modified SA approach.

We find that our flavor of the partitioning problem can be solved using standard graph partitioning packages. Moreover, post-optimization of the partitions obtained through classical graph partitioning packages can be performed well with our SA scheme. As expected, the time to evaluate matrix polynomials for different system (graph) sizes decreases with both the number of processors and parts in our simulations. Our main result is that the increased effort for graph partitioning and post-optimization with SA is beneficial overall when applying our parallelized version of the G-SP2 algorithm to meaningful physical systems. Based on our observation that *METIS* with a SA post-processing step is significantly faster than competing methods while giving the best results on average, we recommend this combination for practical use.

References

Aleksandrov, L. and Djidjev, H. N. (1996). Linear algorithms for partitioning embedded graphs of bounded genus. *SIAM J. Discrete Math.*, 9(1):129–150.

Bader, D. A., Meyerhenke, H., Sanders, P., and Wagner, D., editors (2013). *Graph Partitioning and Graph Clustering - 10th DIMACS Implementation Challenge Workshop*, volume 588 of *Contemporary Mathematics*. AMS.

Bock, N. and Challacombe, M. (2013). An optimized sparse approximate matrix multiply for matrices with decay. *SIAM Journal on Scientific Computing*, 35(1):C72–C98.

Borstnik, U., VandeVondele, J., Weber, V., and Hutter, J. (2014). Sparse matrix multiplication: The distributed block-compressed sparse row library. *Parallel Computing*, 40:47–58.

Bunn, C. (1939). The crystal structure of long-chain normal paraffin hydrocarbons. The "shape" of the CH2 group. *Trans. Faraday Soc.*, 35:482–491.

Djidjev, H. N., Hahn, G., Mniszewski, S. M., Negre, C. F., Niklasson, A. M., and Sardeshmukh, V. (2016). Graph partitioning methods for fast parallel quantum molecular dynamics (full text with appendix). *SIAM Workshop on Combinatorial Scientific Computing (CSC16). arXiv:1605.01118*, pages 1–17.

Elstner, M., Porezag, D., Jungnickel, G., Elsner, J., Haugk, M., Frauenheim, T., Suhai, S., and Seifert, G. (1998). Self-consistent-charge density-functional tight-binding method for simulations of complex materials properties. *Phys. Rev. B*, 58(11):7260–7268.

Fiduccia, C. and Mattheyses, R. (1982). A linear time heuristic for improving network partitions. *In Proc. 19th IEEE Design Automation Conference*, pages 175–181.

Finnis, M. W., Paxton, A. T., Methfessel, M., and van Schilfgarde, M. (1998). Crystal structures of zirconia from first principles and self-consistent tight binding. *Phys. Rev. Lett.*, 81:5149.

Ford Jr., L. and Fulkerson, D. (1956). Maximal flow through a network. *Canad. J. Math.*, 8:399–404.

Frauenheim, T., Seifert, G., aand Z. Hajnal, M. E., Jungnickel, G., Poresag, D., Suhai, S., and Scholz, R. (2000). A self-consistent charge density-functional based tight-binding method for predictive materials simulations in physics, chemistry and biology. *Phys. Stat. sol.*, 217:41.

Hendrickson, B. and Leland, R. (1995). A Multi-Level Algorithm For Partitioning Graphs. *Proceedings of the IEEE/ACM SC95 Conference.*

Karypis, G. and Kumar, V. (1998). A hypergraph partitioning package.

Karypis, G. and Kumar, V. (1999). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392.

Karypis, G. and Kumar, V. (2000). Multilevel k-way hypergraph partitioning. *VLSI Design*, 11(3):285–300.

Kernighan, B. W. and Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *Bell Sys. Tech. J.*, 49(2):291–308.

Kirkpatrick, S., Gelatt Jr, C., and Vecchi, M. (1983). Optimization by simulated annealing. *Science*, 200(4598):671–680.

Lipton, R. J. and Tarjan, R. E. (1979). A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189.

Miller, G. (1997). Separators for sphere-packings and nearest neighbor graphs.

Mniszewski, S. M., Cawkwell, M. J., Wall, M., Moyd-Yusof, J., Bock, N., Germann, T., and Niklasson, A. M. (2015). Efficient parallel linear scaling construction of the density matrix for born-oppenheimer molecular dynamics. *J Chem Theory Comput*, 11(10):4644–4654.

Niklasson, A. M. (2002). Expansion algorithm for the density matrix. *Phys. Rev. B*, 66(15):155115–155121.

Niklasson, A. M., Mniszewski, S. M., Negre, C. F., Cawkwell, M. J., Swart, P. J., Mohd-Yusof, J., Germann, T. C., Wall, M. E., Bock, N., Rubensson, E. H., and Djidjev, H. N. (2016). Graph-based linear scaling electronic structure theory. *J Chem Phys*, 144(23):234101.

Sanders, P. and Schulz, C. (2011). Engineering multilevel graph partitioning algorithms. In *LNCS*, volume 6942, pages 469–480.

Sanders, P. and Schulz, C. (2013). Think Locally, Act Globally: Highly Balanced Graph Partitioning. In *International Symposium on Experimental Algorithms (SEA)*, volume 7933 of *LNCS*, pages 164–175. Springer.

VandeVondele, J., Bortnik, U., and Hutter, J. (2012). Linear scaling self-consistent field calculations with millions of atoms in the condensed phase. *Journal of Chemical Theory and Computation*, 8(10):3565–3573. PMID: 26593003.

Walshaw, C. and Cross, M. (2000). Mesh Partitioning: A Multilevel Balancing and Refinement Algorithm.

Wulff-Nilsen, C. (2011). Separator theorems for minor-free and shallow minor-free graphs with applications. In *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 37–46.

A Proofs for Section 2

For any graph I and vertex $v \in I$, the *neighborhood* of v in I is the set $N(v, I) = \{w \in V(I) \mid (v, w) \in E(I)\}$. Let $H = \mathcal{P}(G)$, v be a vertex of G, and H_v denote the subgraph of H induced by $N(v, H)$. For the following lemmas we assume that $T_i \cap E(H) = \emptyset$ for all i, i.e., none of the edges in $H = \mathcal{P}(G)$ are thresholded. We have the following properties.

Lemma 2. *Let v be a vertex of G. Then $N(v, \mathcal{P}(G)) = N(v, \mathcal{P}(H_v))$.*

Proof. First we prove that $N(v, \mathcal{P}(G)) \subseteq N(v, \mathcal{P}(H_v))$. Let $w \in N(v, \mathcal{P}(G))$. Then $(v, w) \in E(\mathcal{P}(G))$ and hence $(v, w) \in E(H_v) \subseteq E(H)$. Since by assumption $T_i \cap E(H) = \emptyset$, $(v, w) \notin T_i$ for all i. From $(v, w) \in E(H_v)$, the last relation, and the fact that all vertices of H have loops, $(v, w) \in E(\mathcal{P}(H_v))$. Hence, $w \in N(v, \mathcal{P}(H_v))$.

Now we prove that $N(v, \mathcal{P}(H_v)) \subseteq N(v, \mathcal{P}(G))$. Let $w \in N(v, \mathcal{P}(H_v))$. Since $\mathcal{P}(H_v)$ and H_v have the same vertex sets, we have $w \in N(v, H_v)$. Furthermore, since H_v is a subgraph of H, $w \in N(v, H) = N(v, \mathcal{P}(G))$. □

The lemma shows that v has the same neighbors in $\mathcal{P}(G)$ and $\mathcal{P}(H_v)$, i.e., their corresponding matrices have nonzero entries in the same positions in the row (or column) corresponding to v. We will next strengthen that claim by showing that the corresponding nonzero entries contain equal values.

Let X_v be the submatrix of A defined by all rows and columns that correspond to vertices of $V(H_v)$. We will call vertex v the *core* and the remaining vertices the *halo* of $V(H_v)$. We define the set $\{V(H_v) \mid v \in G\}$ to be the *CH-partition* of G. Note that, unlike other definitions of a partition used elsewhere, the vertex sets of CH-partitions (and, specifically, the halos) can be, and typically are, overlapping.

Lemma 3. *For any $v \in V(G)$ and any $w \in N(v, \mathcal{P}(G))$, the element of $P(A)$ corresponding to the edge (v, w) of $\mathcal{P}(G)$ is equal to the element of $P(X_v)$ corresponding to the edge (v, w) of $\mathcal{P}(H_v)$.*

Proof. Let $m = 2^s$ be the degree of P. We will prove the lemma by induction on s. Clearly, the claim is true for $s = 0$ since the elements of both A^1 and X^1 are original elements of the matrix A. Assume the claim is true for $s-1$. Define $P' = P_1 \circ T_1 \circ \ldots \circ P_{s-1} \circ T_{s-1}$. By the inductive assumption, the corresponding elements in the matrices $A' = P'(A)$ and $X' = P'(X)$ have equal values. We need to prove the same for the elements of A'^2 and X'^2.

Let $(v, w) \in E(\mathcal{P}(G))$. By Lemma 2, $(v, w) \in E(\mathcal{P}(H_v))$. For each vertex u of $\mathcal{P}(H_v)$ let u' denote the corresponding row/column of X. We want to show that $P(A)(v, w) = P(X)(v', w')$.

By definition of matrix product, $A'^2(v, w) = \sum A'(v, u)A'(u, w)$, where the summation is over all u such that $(v, u), (u, w) \in E(\mathcal{P}(G))$. Similarly, $X'^2(v', w') = \sum X'(v', u')X'(u', w')$, where the summation is over the values of u' corresponding to the values of u from the previous formula, by Lemma 2. By the inductive assumption, $A'(v, u) = X'(v', u')$ and $A'(u, w) = X'(u', w')$, thus $A'^2(v, w) = X'^2(v', w')$. Since by assumption $A'(v, w) = X'(v', w')$, we have $P_s(A')(v, w) = P_s(X')(v', w')$, and hence $P(A)(v, w) = (P_s \circ T_s)(A')(v, w) = (P_s \circ T_s)(X')(v', w') = P(X)(v', w')$. □

Lemma 3 implies the following algorithm to compute $\mathcal{P}(A)$ given we know its sparsity structure in H_v:

1. Construct a CH-partition Π of G into n parts such that each part consists of a vertex (core) and its adjacent vertices in H_v (halo);

2. For the i-th part Π_i of Π whose core is the i-th vertex of $G(A)$, construct a submatrix A_i containing the rows and columns of A corresponding to the vertices of Π_i;

3. Compute $\mathcal{P}(A_i)$ for all i;

4. Define $\mathcal{P}(A)$ as a matrix whose i-th row has nonzero elements corresponding to the i-th row of $\mathcal{P}(A_i)$ (subject to appropriate reordering).

Clearly, in many cases it will be advantageous to consider CH-partitions whose cores contain multiple vertices. We will next show that the above approach for CH-partitions with single-node cores can be generalized to the multi-node core case.

We will generalize the definitions of $N(v, \mathcal{P}(G))$ and $N(v, \mathcal{P}(H_v))$ for the case where the vertex v is replaced by a set U of vertices of G. For any graph I, we define $N(U, I) = \bigcup_{v \in U} N(v, I)$. Furthermore, we define by H_U the subgraph of H induced by $N(U, H)$.

Suppose the sets $\{U_i \mid i = 1, \ldots, q\}$ are such that $\bigcup_i U_i = V(G(A))$ and $U_i \cap U_j = \emptyset$. In this case we can define a CH-partition of $G = G(A)$ consisting of q sets, where for each i, U_i is the core and $N(U_i, H) \setminus U_i$ is the halo of Π_i.

The following generalizations of Lemma 2 and Lemma 3 follow in a straightforward manner.

Lemma 4. *Denote by H_i the subgraph $\mathcal{P}(H_{U_i})$ of G. Let v be a vertex of U_i. Then $N(v, \mathcal{P}(G)) = N(v, H_i)$.*

Denote by A_{U_i} the submatrix of A consisting of all rows and columns that correspond to vertices of $V(H_{U_i})$. The following main result of this section shows that $P(A)$ can be computed on submatrices of the Hamiltonian: this insight justifies the parallelized evaluation of a matrix polynomial.

Lemma 5. *For any $v \in U_i$ and any $w \in N(v, \mathcal{P}(G))$, the element of $P(A)$ corresponding to the edge (v, w) of $\mathcal{P}(G)$ is equal to the element of $P(A_{U_i})$ corresponding to the edge (v, w) of H_i.*

In case of a QMD simulation, as outlined in Section 2, we assume that the sparsity structure of $P(A)$ can be approximated by the one of the density matrix D from the previous QMD simulation step. For the halos we approximate $H = \mathcal{P}(G)$ with $G(D)$ as before, and use the current H as a substitute for the halos. This leads to the generalization of the partitioned algorithm for computing $P(A)$ in case of multi-vertex cores given at the end of Section 2.

B Further Details on Experimental Results

Raw data for the experiments described in Section 4.3 can be found in Table 2. The six partitioning schemes we use are listed in column "methods". The performance of those methods is measured in four different ways: As a measure of the total matrix multiplication cost of a step of the SP2 algorithm, we report the sum of cubes criterion (3) in column 3 ("sum"). The sum of cubes criterion is also a measure of the computational effort of SP2 since matrix multiplications consume most of the computation time in SP2. As a measure of the variability of partition sizes created by our algorithm, we report the smallest and largest size of any CH-partition (columns 4 ("min") and 5 ("max")). In the best case scenario, the sizes of all partitions should be roughly equal since otherwise, the nodes or processors will have very unequal computational loads in our parallel implementation of the SP2 algorithm. This is undesirable in practice. The average computation time (in seconds) for each partitioning algorithm is shown in the last column ("time").

Table 2: Various test systems (first column) evaluated by different partition schemes (second column; number of vertices n, edges m, and partitions p are given). Results measured with the sum of cubes (3) criterion (sum), the sum of size and halo for the smallest CH-partition (min) as well as the biggest CH-partition (max), and the overall runtime (in seconds) are given in columns 3-6.

Test system	method	sum	min	max	time [s]
polyethylene dense crystal	METIS	57,982,058,496	1536	1536	0.267175
n = 18432	METIS + SA	51,856,752,364	976	1536	0.347209
m = 4112189	HMETIS	7,126,357,377,024	3840	9984	141.426
p = 16	HMETIS + SA	1,362,943,612,944	2520	5814	141.79
	KaHIP	32,614,907,904	768	1536	0.7
	KaHIP + SA	32,614,907,904	768	1536	0.73
polyethylene sparse crystal	METIS	24,461,180,928	1152	1152	0.024942
n = 18432	METIS + SA	24,461,180,928	1152	1152	0.030508
m = 812343	HMETIS	195,689,447,424	2304	2304	55.9726
p = 16	HMETIS + SA	170,056,587,295	2013	2299	55.9943
	KaHIP	24,461,180,928	1152	1152	0.07
	KaHIP + SA	24,461,180,928	1152	1152	0.08
phenyl dendrimer	METIS	336,049,081	150	409	0.13482
n = 730	METIS + SA	146,550,740	0	382	0.14877
m = 31147	HMETIS	177,436,462	135	358	1.578
p = 16	HMETIS + SA	118,409,940	0	358	1.59436
	KaHIP	231,550,645	55	381	1.72
	KaHIP + SA	116,248,715	0	324	1.74
polyalanine 189	METIS	1,305,573,505,507	3358	5145	0.332091
n = 31941	METIS + SA	1,297,206,329,828	3362	5093	0.372463
m = 1879751	HMETIS	1,402,737,703,273	3762	5124	418.229
p = 16	HMETIS + SA	1,393,115,476,879	3765	5119	418.28
	KaHIP	1,649,301,823,304	12	6030	18.35
	KaHIP + SA	1,624,800,725,049	12	5983	18.39
peptide 1aft	METIS	603,251	24	41	0.004755
n = 384	METIS + SA	572,281	24	41	0.005007
m = 1833	HMETIS	562,601	24	40	0.820561
p = 16	HMETIS + SA	538,345	24	42	0.820771
	KaHIP	575,978	11	44	0.08
	KaHIP + SA	575,978	11	44	0.08
polyethylene chain 1024	METIS	8,961,763,376	800	848	0.01513
n = 12288	METIS + SA	8,961,763,376	800	848	0.017951
m = 290816	HMETIS	8,951,619,584	824	824	27.3297
p = 16	HMETIS + SA	8,951,619,584	824	824	27.3332
	KaHIP	9,037,266,968	782	875	0.73
	KaHIP + SA	9,000,224,048	782	872	0.74
polyalanine 289	METIS	2,816,765,783,803	4591	6102	0.366308
n = 41185	METIS + SA	2,816,141,689,603	4591	6093	0.399265
m = 1827256	HMETIS	3,694,884,690,563	5733	6828	710.084
p = 16	HMETIS + SA	3,681,874,557,307	5733	6830	710.128
	KaHIP	4,347,865,055,912	52	8955	43.9
	KaHIP + SA	4,309,969,305,955	52	8907	43.94
peptide trp cage	METIS	35,742,302,607	1228	1414	0.025795
n = 16863	METIS + SA	35,740,265,780	1228	1414	0.029837
m = 176300	HMETIS	35,428,817,730	1214	1472	31.0506
p = 16	HMETIS + SA	35,237,003,004	1214	1472	31.0545
	KaHIP	43,551,196,287	515	1898	2.81
	KaHIP + SA	43,388,946,192	536	1896	2.81
urea crystal	METIS	4,126,744,977	608	708	0.047032
n = 3584	METIS + SA	4,126,744,977	608	708	0.057645
m = 109067	HMETIS	5,913,680,136	643	811	15.2321
p = 16	HMETIS + SA	5,194,749,106	604	785	15.2443
	KaHIP	3,907,671,473	622	630	1.05
	KaHIP + SA	3,907,671,473	622	630	1.05