

Parallel adaptive integration in high-performance functional Renormalization Group computations

Julian Lichtenstein¹, Jan Winkelmann², David Sánchez de la Peña¹, Toni Vidović³, and Edoardo Di Napoli^{2,4}

¹ Institute for Theoretical Solid State Physics, RWTH Aachen University.

² Aachen Institute for Advance Study in Computational Engineering Science.

³ Department of Mathematics, Faculty of Science, University of Zagreb.

⁴ Jülich Supercomputing Centre, Forschungszentrum Jülich.

Abstract. The conceptual framework provided by the functional Renormalization Group (fRG) has become a formidable tool to study correlated electron systems on lattices which, in turn, provided great insights to our understanding of complex many-body phenomena, such as high-temperature superconductivity or topological states of matter. In this work we present one of the latest realizations of fRG which makes use of an adaptive numerical quadrature scheme specifically tailored to the described fRG scheme. The final result is an increase in performance thanks to improved parallelism and scalability.

Keywords: adaptive quadrature, functional Renormalization Group, interacting fermions, hybrid parallelization, shared memory parallelism

1 Introduction

In this paper we report on the algorithmic and performance improvements resulting from the collaboration between High-Performance Computing (HPC) experts and domain scientists in the specific field of functional Renormalization Group (fRG). In particular, we focus on an adaptive implementation of a two-dimensional numerical quadrature algorithm tailored to the evaluation of a large number of integrals within a recently developed fRG method. The result of such an effort is the Parallel Adaptive Integration in two Dimensions (PAID) library. PAID requires approximately an order of magnitude fewer operations for the computation of the numerical integrals and translates this reduction into a substantial gain in parallel performance.

The Renormalization Group (RG) is a powerful method describing the behavior of a physical system at different energy and length scales. RG techniques allow smooth interpolation between well studied models at a given energy scale and complicated emergent phenomena at lower energy scales. In what is known as *RG flow*, physical quantities are computed iteratively with respect to variation of an auxiliary scale parameter by solving a system of coupled ordinary integro-differential equations. In its application to interacting electrons on a lattice at low temperatures, fRG methods are commonly used to detect transitions

of the metallic state towards some ordered state [1, 2]. At the initial energy scale of the flow, the physical system is in a well understood metallic state, where weakly correlated electrons interact pairwise through Coulomb repulsion. Lowering the scale, a second order phase transition may take place at some critical temperature, in which some form of order (e.g. magnetism or superconductivity) spontaneously emerges.

fRG methods passed through many refinements over the years [3–6] in the form of specific approximation schemes. While each of these schemes has its strength, the improvement of their accuracy (e.g. on predictions for critical temperatures) is still underway. In the current work, we illustrate the *Truncated Unity* scheme (TUfRG) [7] and its parallel implementation. One of the computational advantages of this scheme stems from the insertion of truncated partitions of unity in the flow equations. The resulting numerical integration becomes less challenging at the expense of having extra operations to perform (so-called inter-channel projections). At each step of the equations’ flow one ends up computing multiple independent integrals parametrized by three indices, namely l , m , and n . In the original TUfRG code all these integrals are distributed over a large number of threads where each one is computed sequentially by a single thread using the adaptive DCUHRE library [8]. Since computing such integrals accounts for around 80% of the total computational time, they are the ideal candidate for an HPC optimization.

While recent implementations have shown increased performance and scalability through parallel quadrature schemes [9, 10], we follow a different path by tailoring the numerical quadrature to the needs of the TUfRG algorithm. In PAID, the subset of all integrals corresponding to one value of l are collected in a container and computed adaptively. All the integrals in the container become tasks which are executed under just one parallel region over the shared memory of a compute node. With this approach we intend to gain better control over the global quadrature error and minimize load imbalance while increasing scalability. Our results show that PAID scales as well as the trivial parallelization using DCUHRE. In addition, PAID’s adaptivity over the indexes m and n of the integrals consistently yields a speedup from $2\times$ up to $4\times$. In section 2 we give a brief account of the method at the base of the TUfRG scheme. In the following section we present the basic notion of adaptive integration and the algorithm underlying the PAID library. In section 4 we describe the parallel implementation in more detail. We conclude with a section on numerical results and future work.

2 The fRG method and the Truncated Unity approach

In this section we provide a short introduction to the mathematical framework of the TUfRG scheme. This is by no means an exhaustive description and we refer the reader to [7] for a detailed presentation. As several other fRG methods,

TUfRG focuses on interacting electrons on 2D lattice systems. These systems exhibit strong correlations at low energy, which results in a rich diversity of ordered ground states. At the mathematical level, the effective two-particle coupling function $V(k_{0,1}, \mathbf{k}_1; k_{0,2}, \mathbf{k}_2; k_{0,3}, \mathbf{k}_3)$ contains essential information on the properties of the electronic ground state. V depends on both three frequencies (k_0) and three momenta ($\mathbf{k}$), while the fourth ones are fixed due to conservation of energy and momentum respectively. In favor of a short notation, we sum up the dependence on frequency and momentum of one particle into a combined index $k = (k_0, \mathbf{k})$ and write $V(k_1, k_2, k_3)$.

The fRG calculation is based on the insertion of a control parameter Ω , which is an artificial energy scale. It is used for tuning the system from an easily solvable starting point to a system that includes the physically important features. For a given value of the control parameter, the strong correlations at energies below Ω are excluded from V . Starting from a high enough Ω results in a well-defined initial value for V , which corresponds to an interaction between two isolated charges. By decreasing Ω , we successively include correlation effects into the effective two-particle coupling. From a mathematical point of view, the calculation of V at lower energy scales can be seen as an *initial value problem*, where the value of V at a high energy scale is the initial value. In order to obtain the resulting two-particle coupling function at lower values of Ω , one needs to integrate a first order ordinary differential equation extracted from a so-called level-2 truncation of the fRG equation hierarchy [1] and from neglecting self-energies. Such an equation can be written as

$$\dot{V}(k_1, k_2, k_3) = \mathcal{T}_{\text{pp}}(k_1, k_2, k_3) + \mathcal{T}_{\text{ph}}^{\text{cr}}(k_1, k_2, k_3) + \mathcal{T}_{\text{ph}}^{\text{d}}(k_1, k_2, k_3), \quad (1)$$

where the dependence on Ω of all quantities is implicit and the dot above V denotes the first derivative with respect to the artificial energy scale. The right-hand side is divided in three main contributions: a particle-particle

$$\mathcal{T}_{\text{pp}} = - \int dp [\partial_{\Omega} G(p) G(k_1 + k_2 - p)] V(k_1, k_2, p) V(k_1 + k_2 - p, p, k_3), \quad (2)$$

a crossed particle-hole

$$\mathcal{T}_{\text{ph}}^{\text{cr}} = - \int dp [\partial_{\Omega} G(p) G(p + k_3 - k_1)] V(k_1, p + k_3 - k_1, k_3) V(p, k_2, p + k_3 - k_1) \quad (3)$$

and three direct particle-hole terms summarized in $\mathcal{T}_{\text{ph}}^{\text{d}}$ as

$$\begin{aligned} \mathcal{T}_{\text{ph}}^{\text{d}} = & \int dp [\partial_{\Omega} G(p) G(p + k_2 - k_3)] [2V(k_1, p + k_2 - k_3, p) V(p, k_2, k_3) \\ & - V(k_1, p + k_2 - k_3, k_1 + k_2 - k_3) V(p, k_2, k_3) \\ & - V(k_1, p + k_2 - k_3, p) V(p, k_2, p + k_2 - k_3)] . \end{aligned} \quad (4)$$

All five summands that appear as integrands are quadratic in both V and the function $G(k) = \frac{\theta(k)}{ik_0 - \epsilon(\mathbf{k})}$, which is the propagator of the system containing

non-interacting particles. The regulator function θ implements the exclusion of correlation effects from the system at energies below Ω . In this paper we use $\theta(k) = \theta(k_0) = \frac{k_0^2}{k_0^2 + \Omega^2}$ as regulator, which suppresses G for Ω much larger than all relevant energy scales of the system. In the limit of $\Omega \rightarrow 0$ the structure of G is recovered and we regain the physical system. The energy dispersion $\epsilon(\mathbf{k})$ —which appears in the denominator of G —contains the energy spectrum of the single-particle problem. Since in this paper we are dealing with a t - t' Hubbard model on a square lattice, the dispersion is

$$\epsilon(\mathbf{k}) = -2t(\cos(k_x) + \cos(k_y)) - 4t' \cos(k_x) \cos(k_y) - \mu \quad (5)$$

where t and t' describe the kinetics of the particles and μ is the chemical potential controlling the total number of particles in the system.

The calculation of the two-particle coupling V using a direct implementation of Eqs. (1)–(4) is numerically challenging. Even if the dependence on the external frequencies $k_{0,1}$, $k_{0,2}$ and $k_{0,3}$ is neglected—as we will do in the following—the scaling of the number of differential equations with respect to the number of momentum sampling points is cubic. Using frequency independent two-particle couplings, the frequency integrals from Eqs. (2)–(4) involve just the Ω derivative of a product of two fermionic propagators and can be performed analytically. The result of this shows sharp structures as function of momentum at small values of Ω (see Fig. 1). As mentioned above, at low energy scales the system can get close to a phase transition, which is indicated by a strong increase of specific components of V . Thus a product of two strongly peaked two-particle couplings and sharp structured propagators constitutes the integrands of the momentum integrals in Eqs. (2)–(4).

In order to change Eqs. (1)–(4) in the direction of a numerically easier treatment, accompanied by the introduction of quantities with a more direct physical interpretation, we perform modifications that can be classified in three steps.⁵ First, the initial part $V^{(0)}$ is separated from the two-particle coupling and the rest is split into three single-channel coupling functions Φ^P , Φ^C and Φ^D . Their derivatives with respect to Ω are given by $\mathcal{T}_{pp}$, $\mathcal{T}_{ph}^{cr}$ and $\mathcal{T}_{ph}^d$ respectively. This is motivated by the fact that V develops strong dependencies on the external momentum combinations appearing in Eqs. (2)–(4) respectively, denoted by $\mathbf{l}$ in the following. In a second step, the remaining weak momentum dependencies of each channel are expanded in a complete set of orthonormal functions $\{f_n\}$ —so-called form-factors. Since we can only use a finite number of basis functions while doing numerics, we restrict the basis to slowly oscillating functions to achieve a good description of weak momentum dependencies of the channels. This latter step can be interpreted as a sort of discretization with $\Phi_{\mathbf{l},\mathbf{k},\mathbf{k}'}^P \rightarrow P_{m,n}(\mathbf{l})$ —and equivalently for the C and D channels—where $\mathbf{k}$ and $\mathbf{k}'$ are replaced by form-factor indices m and n . As a consequence of implementing the first two steps in Eqs. (1)–(4), the scaling of the number of coupled differential equations respect to the number of momentum sampling points is reduced to a linear relation.

⁵ See Ref. [7] for a more detailed derivation and an example application of the scheme.

Moreover, the scaling respect to the number of form-factors is less important in most cases, since a good description can be achieved even using a small number of form-factors.

In a third and final step we change the form of the RHS of the resulting differential equations by inserting two partitions of unity of the form-factor basis set. The fermionic propagators can then be separated from the two-particle coupling terms and the differential equation (1) now takes the form of three separate equations

$$\dot{\mathbf{P}}(\mathbf{l}) = \mathbf{V}^P(\mathbf{l}) \dot{\mathcal{X}}^{\text{pp}}(\mathbf{l}) \mathbf{V}^P(\mathbf{l}), \quad (6)$$

$$\dot{\mathbf{C}}(\mathbf{l}) = -\mathbf{V}^C(\mathbf{l}) \dot{\mathcal{X}}^{\text{ph}}(\mathbf{l}) \mathbf{V}^C(\mathbf{l}), \quad (7)$$

$$\dot{\mathbf{D}}(\mathbf{l}) = 2\mathbf{V}^D(\mathbf{l}) \dot{\mathcal{X}}^{\text{ph}}(\mathbf{l}) \mathbf{V}^D(\mathbf{l}) - \mathbf{V}^C(\mathbf{l}) \dot{\mathcal{X}}^{\text{ph}}(\mathbf{l}) \mathbf{V}^D(\mathbf{l}) - \mathbf{V}^D(\mathbf{l}) \dot{\mathcal{X}}^{\text{ph}}(\mathbf{l}) \mathbf{V}^C(\mathbf{l}), \quad (8)$$

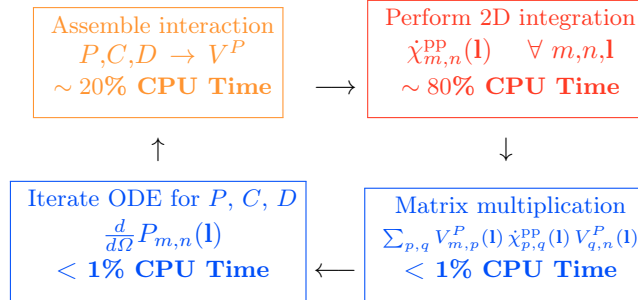
where

$$\mathcal{X}_{m,n}^{\text{pp}}(\mathbf{l}) = \int d\mathbf{p} \left[\int dp_0 G\left(p_0, \frac{1}{2} + \mathbf{p}\right) G\left(-p_0, \frac{1}{2} - \mathbf{p}\right) \right] f_m(\mathbf{p}) f_n(\mathbf{p}), \quad (9)$$

$$\mathcal{X}_{m,n}^{\text{ph}}(\mathbf{l}) = \int d\mathbf{p} \left[\int dp_0 G\left(p_0, \mathbf{p} + \frac{1}{2}\right) G\left(p_0, \mathbf{p} - \frac{1}{2}\right) \right] f_m(\mathbf{p}) f_n(\mathbf{p}). \quad (10)$$

$\mathbf{V}^P$, $\mathbf{V}^C$ and $\mathbf{V}^D$ are two-particle couplings with two momenta replaced by form-factor indices, and can be computed from P , C , and D in the aforementioned inter-channel projections. The inserted partitions of unity are also truncated by ignoring strongly oscillating form-factors. Inner integrals from Eqs. (9) and (10) can be treated analytically, while the calculation of the outer (momentum) integrals requires a sophisticated numerical integration scheme. Due to the last modification, we call the scheme described in Eqs. (6)–(10) *Truncated Unity fRG* (TUFfRG).

Numerically, this scheme is implemented in four steps organized in a loop mimicking the flow of the ODE for decreasing values of Ω . Within the loop, the most intensive part of the computation is given by the numerical integration. In the current C++ implementation of TUFfRG, the numerical integration is parallelized using the MPI+OpenMP paradigm. Each MPI process receives a subset of values of $\mathbf{l}$ indices while an OpenMP `parallel for` pragma encapsulates the actual computation of the integrals for all m and n values. Each integral is then assigned to a thread and computed sequentially using the DCUHRE library [8].



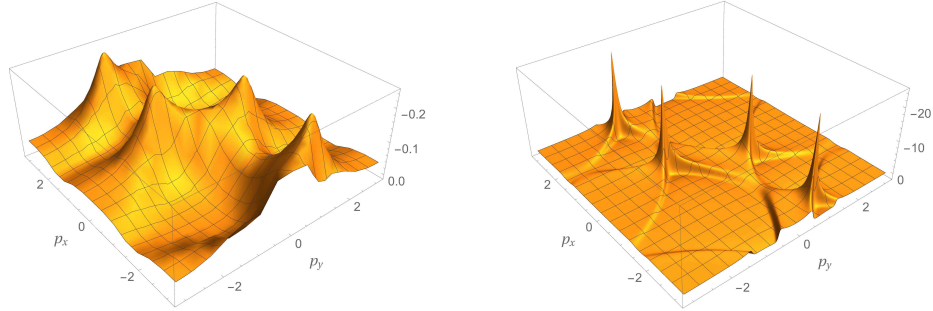


Fig. 1. The value of the integrand is plotted against the two-dimensional momentum $\mathbf{p}$ for $\Omega = 1.0$ (left plot) and $\Omega = 0.1$ (right plot). In this example case the external momentum $\mathbf{l}$ is set to $(3.14, 0.78)$ and both form-factor indices label the lowest order function, which is constant in momentum space.

Using example values of $\mathbf{l}$, m and n , Fig. 1 shows the integrand from Eq. (9) at a high and a low value of Ω . This example illustrates a general characteristic of the integrands: While at high Ω values the variations in momentum space are smooth, sharp edges and peaks emerge as the energy scale is lowered. This means in terms of numerical integration that the density of sampling points in momentum space should be chosen adaptively and separately for every integration. As the data from Fig. 1 suggest, the adaptive routine used should furthermore be able to refine the grid of sampling points using strongly local criteria in order to reduce the inaccuracies caused by sharp structures and to save time when flat regions are considered. In the next section we show how such a target is achieved by Algorithm 2, and give an account of its parallel implementation.

3 Adaptive integration a-la Clenshaw-Curtis

The main target of adaptive integration is to decrease the error in a consistent and controlled fashion over relatively low-dimensional domains [11]. A possible choice to increase accuracy is to increase the number of integration points for a given integration method. Alternatively one can fix the number of integration points and instead partition the integration domain. Numerical integration with the same number of integration points is then performed on each subdomain. The latter method is known as *adaptive integration*. One of the most accurate variations of such a method computes the error across the whole domain [12, Chapter 6]. If some estimate for the global error is above a given threshold, one iteratively subdivides and integrates the sub-domain with the largest local error. Algorithm 1 shows the typical structure of such an adaptive integration scheme.

Algorithm 1 Adaptive integration on domain $\mathcal{D}$ with global error criterion

```

function ADAPTIVE(Integrand  $\phi$ , Domain  $\mathcal{D}$ , Target Error  $\varepsilon$ )
  Compute integrals  $Q(\phi, \mathcal{D}, n_1)$  and  $Q(\phi, \mathcal{D}, n_2)$ 
   $ERR[\phi] = |Q_{n_1} \phi - Q_{n_2} \phi|$ 
  Store domain  $\mathcal{D}$  and  $ERR[\phi]$ 
  while  $ERR[\phi] > \varepsilon$  do
    Take the sub-domain  $\mathcal{D}^s$  with largest error
    Subdivide it into parts  $\mathcal{D}^{s_0}, \dots, \mathcal{D}^{s_{d-1}}$ 
    for  $a = 1 : d$  do
      Compute  $Q(\phi, \mathcal{D}^{s_a}, n_1)$  and  $Q(\phi, \mathcal{D}^{s_a}, n_2)$ 
       $ERR^{s_a}[\phi] = |Q_{n_1}^{s_a} \phi - Q_{n_2}^{s_a} \phi|$ 
      Store domain  $\mathcal{D}^{s_a}$  and  $ERR^{s_a}[\phi]$ 
    end for
     $ERR[\phi] = \sum_s ERR^s[\phi]$ 
  end while
  return  $VALUE = \sum_s Q(\phi, \mathcal{D}^s, n_2)$ 
end function

```

In the current work we compute the numerical integrals (and also the estimate for the error) using the Clenshaw-Curtis Quadrature (CCQ) formula⁶. Starting with n_1 integration points, we settle for a formula with $n_2 \geq n_1$ as the more accurate estimate [14]. With this choice the error estimate is equal to

$$ERR[\phi] = |Q_{n_1} \phi - Q_{n_2} \phi|,$$

where with $Q_n \phi = Q(\phi, \mathcal{D}, n)$ we indicate the computation of the integral $\Phi = \int_{\mathcal{D}} \phi$ over the domain $\mathcal{D}$ through numerical quadrature with n integration points⁷. When n_2 is proportional to n_1 , the advantage of the CCQ scheme—compared to Gauss for instance—is the reuse of the n_1 points as a subset of the n_2 points. In the rest of this work we set $n_1 = N$ and $n_2 = 2n_1$.

The schematic description in Algorithm 1 should be applied to the computations of the integral on the RHS of the flow Eqs. (2)–(4). After discretizing and projecting (using a truncated partition of unity), the RHS of such equations are split in multiplications of two-particle couplings $\mathbf{V}^i$ ($i = P, C, D$) and susceptibility factors $\mathcal{X}^j$ ($j = \text{pp, ph}$), only the latter expressed in terms of integrals. Despite the fact that now the integrals seem limited to the RHS of Eqs. (9)–(10), the adaptive approach has to encompass the whole set of integrals labeled by the indices m and n . Moreover, the original integrals included also the values of the couplings $\mathbf{V}^i$ in the integrand, so these quantities play an active role in

⁶ For a review of Clenshaw-Curtis and a comparison with Gauss quadrature rules we refer to the excellent review [13]

⁷ We use the conventional notation indicating with the capital symbol the integral (Φ) and with the corresponding small cap symbol (ϕ) its integrand.

the computation of the global error. Let us clarify this point by considering just the particle-particle channel Eq. (2), and formally evaluating it using a generic quadrature formula

$$Q_N \tau_{pp} = - \sum_{\ell=1}^N w_\ell [\partial_\Omega G(p_\ell) G(k_1 + k_2 - p_\ell)] V(k_1, k_2, p_\ell) V(k_1 + k_2 - p_\ell, p_\ell, k_3), \quad (11)$$

where the w_ℓ are the weights associated with the quadrature points p_ℓ . After some rearrangements and the introduction of the truncated partition of unity, the RHS of this equation is transformed into $\sum_{p,q} V_{m,p}^P (Q_N \dot{\chi}_{p,q}^{pp}) V_{q,n}^P$ where we made explicit the m and n indices and suppressed, for the moment, the dependence on the $\mathbf{l}$ index. Despite the fact that now this quantity is the sum of distinct quadratures $Q_N \dot{\chi}_{p,q}^{pp}$, the global error should be thought as defined by the original expression $|Q_N \tau_{pp} - Q_{2N} \tau_{pp}|$, leading to the following expression

$$\begin{aligned} \text{ERR}[\dot{P}_{m,n}] &= \left| \sum_{p,q} [V_{m,p}^P (Q_N \dot{\chi}_{p,q}^{pp}) V_{q,n}^P - V_{m,p}^P (Q_{2N} \dot{\chi}_{p,q}^{pp}) V_{q,n}^P] \right| \\ &\leq \|V_{m,:}^P\|_\infty \|V_{:,n}^P\|_\infty \sum_{p,q} |Q_N \dot{\chi}_{p,q}^{pp} - Q_{2N} \dot{\chi}_{p,q}^{pp}|, \end{aligned} \quad (12)$$

where $x = V_{m,:}$ is the vector made by all column entries corresponding to the m^{th} row and $\|x\|_\infty = \max_j |x_j|$.

Algorithm 2 Parallel adaptive integration of TUFrg with global error

```

1: for all  $i, j, k, \mathbf{l}$  do
2:   function ADAPTIVE(Integrand  $\phi^{i,j,k}$ , Domain  $\mathcal{D}$ , Target Error  $\varepsilon$ )
3:     done = false
4:     while done  $\neq$  true do
5:       Take the domain  $\mathcal{D}^s \subseteq \mathcal{D}$  and indices  $(p, q)$  with largest error
6:       Subdivide it into parts  $\mathcal{D}^{s_0}, \dots, \mathcal{D}^{s_3}$ 
7:       for  $a = 1 : 4$  do
8:         Compute  $Q(\chi_{p,q}^j, \mathcal{D}^{s_a}, N)$  and  $Q(\chi_{p,q}^j, \mathcal{D}^{s_a}, 2N)$ 
9:          $\text{ERR}^{s_a}[\chi_{p,q}^j] = |Q_N^{s_a} \chi_{p,q}^j - Q_{2N}^{s_a} \chi_{p,q}^j|$ 
10:        Store domain  $\mathcal{D}^{s_a}$ , indices  $(p, q)$  and  $\text{ERR}^{s_a}[\chi_{p,q}^j]$ 
11:      end for
12:       $\text{ERR}[\phi^{i,j,k}] = \sum_{s,p,q} \text{ERR}^s[\chi_{p,q}^j]$ 
13:      if  $\text{ERR}[\phi^{i,j,k}] < \varepsilon$  then done = true end if
14:    end while
15:    return  $(\text{VALUE})_{p,q}^j = \sum_s Q(\chi_{p,q}^j, \mathcal{D}^s, 2N)$ 
16:  end function
17: end for

```

We can think of the entire numerical integration as the union of the quadratures $Q_n \dot{\chi}_{p,q}^{pp}$ on the same domain $\mathcal{D}$ for each value of the indices p and q . While each adaptive quadrature labeled by p and q returns its own `VALUE`, the absolute error is computed globally over all indices (p, q) . We further simplify the definition of the error by dropping the terms proportional to $\|V\|_\infty$. This last step may seem arbitrary but it is in part justified by the fact that, in the actual computation, we are only interested in the error relative to the value of the function. In order to maintain generality we define with $\phi^{i,j,k} = \mathbf{V}^i \dot{\chi}^j \mathbf{V}^k$ and the associated global relative error as

$$\text{ERR}[\phi^{i,j,k}] = \sum_{p,q} \text{ERR}[\dot{\chi}_{p,q}^j].$$

We kept the index `l` still implicit so as to avoid cluttering the notation, but it is understood that all definitions above depend implicitly on it. With these definitions in mind we end up with the adaptive quadrature scheme illustrated in Algorithm 2.

4 Parallel implementation

In this work, we describe a parallel implementation of the `ADAPTIVE` function of Algorithm 2 over one computing node using OpenMP pragmas, and leave the outer `for` loops—identified by indices i , j , k , and `l`—distributed over MPI processes. Each elementary integration is encoded as a `task`, which can be imagined as a struct type. Each `task` has the following members: an `id` field that corresponds to distinct values of the p and q indices, a `domain`, the two values `val_N` and `val_2N` computed according to the CCQ method, and an estimate of the error `err`.

The adaptive integration scheme requires the tasks with the largest error to be scheduled first. Such an approach is not easily expressible with the OpenMP task construct. Although OpenMP tasks have recently gained support for task priorities, the allowed priority values are limited to non-negative scalars. As a result `PAID` cannot make use of OpenMP tasks.

The container into which the tasks are placed is a heap data structure that uses the `err` as the sorting key. A heap structure allows cheap en- and dequeuing of tasks. The heap is initialized at the beginning of the program.

Listing 1. Initialization of the task queue container

```

1  ERR[φ] := 0.0
2  for all (p, q)
3      task.id := (p, q) and task.domain := D
4      task.val_N := Q_N χ and task.val_2N := Q_{2N} χ
5      task.err := |Q_N χ - Q_{2N} χ|
6      container.PUSH(task)
7      ERR[φ] += task.err
8  HEAPIFY(container, key = task.err)
```

The heap structure of the container guarantees that task extraction is done in a way that refines regions with larger error estimates first, independent of which pair of indices (p, q) they belong to. In a way this algorithm can be seen as adaptive integration with starting regions defined by both $\mathcal{D}$ and (p, q) . Due to the adaptivity based on the global error, the OpenMP parallel block has to enclose the domain $\mathcal{D}$ as well as the indices (p, q) . As previously stated, PAID cannot make use of OpenMP's more advanced work sharing constructs. Instead, Algorithm 2 parallelizes the main part of the routine from line 4 to 14 using just the `parallel` directive (see line 4 of Listing 5). Access to the queue in lines 5 and 10 requires exclusive access in order to avoid race conditions. For queues that are not thread-safe a mutex is required (`critical` directive). This may decrease parallel performance as threads may need to wait for access to the queue. For this reason we implement bulk extraction and insertion into a thread-local container: Each thread can extract a `MaxTask` number of tasks, whose value is set by the user. Care has to be taken in choosing `MaxTask`; Its optimal value is a trade-off between maintaining acceptable levels of parallel performance and avoiding unnecessary adaptive refinements.

Listing 2. Extract tasks with maximal error from the queue

```

1  #PRAGMA OMP CRITICAL {
2  for n = 1 : MaxTask
3      local_container[n] = EXTRACT-MAX(container)
4  }
```

This results in a work sharing construct, as each task returned from the heap is different. Tasks are processed by partitioning their domain once in each dimension, which yields four new tasks. Before the new tasks can be inserted into the heap an error estimate is required, which in turn requires evaluation of the integrals.

Listing 3. Divide domains and evaluate new tasks

```

1  for n = 1 : MaxTask
2      evaltask[n, 1 : 4] := Divide local_container[n].domain into 4 parts
3      for a = 1 : 4
4          evaltask[n, a].domain := part a of local_container[n].domain
5          Compute evaltask[n, a].val_N and evaltask[n, a].val_2N
6          Compute evaltask[n, a].err
```

Eventually, the global error is updated within the mutex. Each thread then inserts the new tasks, together with their relative sub-domain, and `id` in the heap.

Listing 4. Update error and insert new tasks in the queue

```

1  #PRAGMA OMP CRITICAL {
2  for n = 1 : MaxTask
3      ERR[φ] -= local_container[n].err
4      for a = 1 : 4
5          ERR[φ] += evaltask[n, a].err
6          INSERT(evaltask[n, a] ⇒ container)
7  }
```

The termination criterion need only be checked by a single thread at the end of its block of refinements. This implies that the termination criterion is checked not sooner than after `MaxTask` refinements. When the global error is lower than the required threshold, all other threads are instructed to exit the while loop via the `done` flag. The entire program, which includes all previous Listings, is illustrated in Listing 5.

Listing 5. Full program

```

1 Program PAID( $\phi, \mathcal{D}, \varepsilon$ )
2   done := false
3   Initialize the task queue container (Listing 1)
4   #PRAGMA OMP PARALLEL {
5     while done  $\neq$  true do
6       Extract tasks with max error from container (Listing 2)
7       Divide the domain and evaluate new tasks (Listing 3)
8       Update ERR[ $\phi$ ] and insert new tasks in the queue (Listing 4)
9       #PRAGMA OMP MASTER {
10        if ERR[ $\phi$ ] <  $\varepsilon$  then done := true
11      }
12    }
13    forall distinct task.id = (p, q)
14      return (value)(p,q) :=  $\sum_{\text{task.domain}}$  task.val_2N

```

5 Results and conclusions

In order to illustrate the effectiveness of PAID within the TUfRG code, we present a number of numerical tests, run on the JURECA computing cluster located at the Jülich Supercomputing Centre. Each node of the cluster is equipped with 2 Intel Xeon E5-2680 v3 Haswell CPUs. All tests were run with a single MPI rank per compute node. Node level parallelism is exclusively due to the shared memory parallelization of the adaptive quadrature implementation described in Algorithm 2.

In the following we draw a comparison between the previous implementation using DCUHRE and the newly developed implementation based on PAID. As both adaptivity and parallel efficiency play an important role in terms of performance, we conducted the comparative analysis in terms of these two aspects separately, before we compare the runtimes.

Fig. 2 shows that the number of function evaluations needed by PAID is smaller than the one needed by DCUHRE at all values of the scale parameter, especially at low scales where most of the computation time is used.⁸ As a smaller number of evaluations implies a more efficient partition of the integration domain, this number can be seen as an inverse measure for the adaptivity of the implementation provided by PAID, especially at low scales where the integrands tend to blow up. The difference in adaptivity between the both schemes can be understood as a consequence of the error estimation. DCUHRE computes the

⁸ Notice that the fRG flow in the current setup starts at high Ω values and successively reduces this scale during the flow.

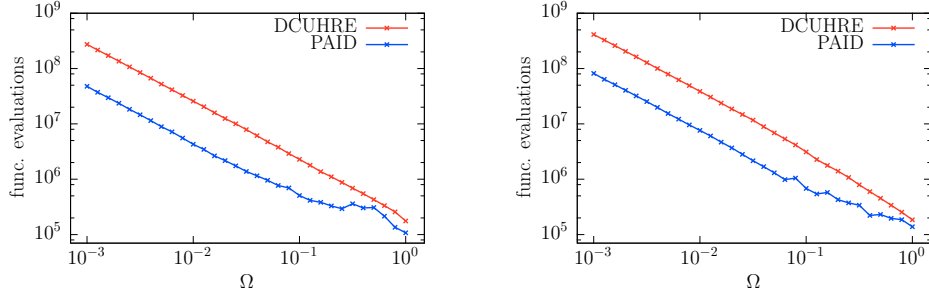


Fig. 2. The number of function evaluations needed for calculating all integrals of a fixed $\mathbf{l}$ value is plotted against Ω . We use a form-factor expansion that results in 45 independent integrals for each external momentum $\mathbf{l}$, which is fixed to $(1.57, 1.31)$ (left plot) and $(2.88, 0.26)$ (right plot) respectively. The results of both implementations—the one using DCUHRE (red) and the one using PAID (blue)—are shown in the same plot in favor of a direct comparison. In order to use the same number of evaluations per subregion as in DCUHRE, we set the PAID parameter N to 4. Further we use $\text{MaxTask} = 10$.

errors of the integrals labeled by p and q in isolation so that each error fulfills the same termination criterion independently. The PAID scheme treats all integrals as *one* task which in practice prioritizes computation over the most difficult partitions of the domain. As the major part of the computation time is used by the low- Ω integration domains, a code that is more efficient in this region of the parameter space pays off in terms of total runtime.

The second performance analysis addresses the parallel efficiency of PAID. Fig. 3 shows that the speedup is close to ideal for any thread number up to 24^9 . Using SMT—up to 48 threads—still increases the speedup compared to the one using 24, but the curves in Fig. 3 show a slower increase in performance. This result suggests that the code is compute bound and can not profit highly from a larger memory bandwidth per core. Although the shared memory parallelization of the implementation of TUFGR using DCUHRE is much simpler—as described in section 2—we find a speedup which is as high as the one we achieve using PAID. We verified that the runtimes of the integrals for distinct p and q within a fixed value of $\mathbf{l}$ do not vary much in serial execution. In such a case the parallelization over the p and q values does not suffer from load imbalances and results in a close to ideal speedup.

In a third set of tests—possibly the most relevant to a user of TUFGR—we compared the runtimes that are needed by DCUHRE and PAID to perform all the integrations within a fixed value of $\mathbf{l}$. As illustrated in Fig. 4, PAID needs less compute time than DCUHRE at all scales and is about 2–3 times faster at low Ω values.

In conclusion the TUFGR code greatly benefits from the proposed adaptive integration algorithm both in terms of load balancing and adaptivity. The re-

⁹ The granularity of the affinity is set to 'compact,core,1'.

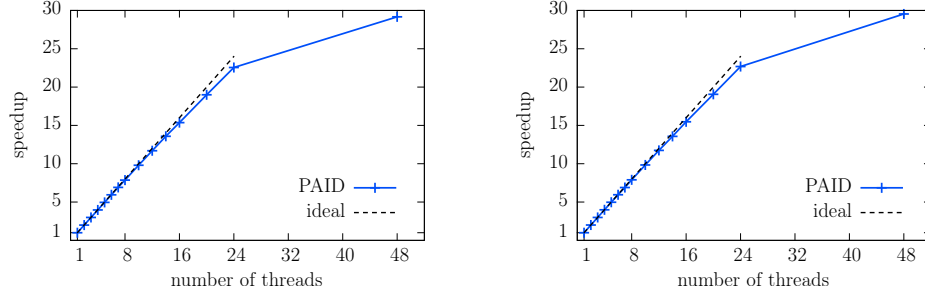


Fig. 3. These plots show the speedup at $\Omega = 10^{-3}$ against the number of threads for the implementation based on PAID. For thread numbers up to 24 each compute core executes only a single thread. At 48 threads each compute core processes two threads at a time using simultaneous multithreading. The $\mathbf{l}$ -values are chosen as in Fig. 2 and there are 325 integrals to calculate per $\mathbf{l}$. For this analysis we use the PAID parameters that result in the best performance: $N = 6$ and $\text{MaxTask} = 18$.

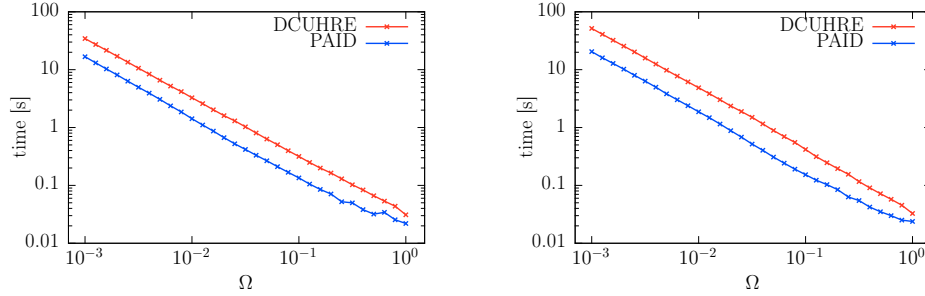


Fig. 4. The computation time needed for calculating all integrals of a fixed $\mathbf{l}$ value using 48 threads is plotted against Ω . We use a form-factor expansion that results in 325 independent integrals for each $\mathbf{l}$, which takes the same values as in Fig. 2. For reasons of comparison we use $N = 4$ in PAID as in the analysis shown in Fig. 2. Further, MaxTask is set to 7.

sult is a good exploitation of the node-level parallelism at any stage of the flow equation without the need of ad-hoc parameter choices. Comparing the new integration scheme with the one provided by DCUHRE shows that the PAID algorithm exhibits a higher level of adaptivity which in turn leads to shorter run-times. In addition, the use of standard OpenMP pragmas ensures performance portability over clusters other than JURECA with the potential for off-loading to many-cores platforms with minimal effort. In the future, we envision to expand the internal parallelism of PAID to distributed memory. Such an extension could replace the currently used distribution of the $\mathbf{l}$ values over the MPI ranks and would prevent load imbalances that limit the number of accessible nodes in the current implementation.

Acknowledgements

Financial support from the Jülich Aachen Research Alliance–High Performance Computing, and the Deutsche Forschungsgemeinschaft (DFG) through grants GSC 111, RTG 1995 and SPP 1459 is gratefully acknowledged. We are thankful to the Jülich Supercomputing Centre (JSC) for the computing time made available to perform the numerical tests. Special thanks to JSC Guest Student Programme which sponsored the research internship of one of the authors.

References

1. Metzner, W., Salmhofer, M., Honerkamp, C., Meden, V., Schönhammer, K.: Functional renormalization group approach to correlated fermion systems. *Rev. Mod. Phys.* **84** (Mar 2012) 299–352
2. Platt, C., Hanke, W., Thomale, R.: Functional renormalization group for multi-orbital fermi surface instabilities. *Advances in Physics* **62**(4-6) (2013) 453–562
3. Zanchi, D., Schulz, H.J.: Weakly correlated electrons on a square lattice: A renormalization group theory. *EPL (Europhysics Letters)* **44**(2) (1998) 235
4. Salmhofer, M., Honerkamp, C.: Fermionic renormalization group flows: Technique and theory. *Progress of Theoretical Physics* **105**(1) (2001) 1–35
5. Husemann, C., Salmhofer, M.: Efficient parametrization of the vertex function, Ω scheme, and the t, t' hubbard model at van hove filling. *Phys. Rev. B* **79** (May 2009) 195125
6. Wang, W.S., Xiang, Y.Y., Wang, Q.H., Wang, F., Yang, F., Lee, D.H.: Functional renormalization group and variational monte carlo studies of the electronic instabilities in graphene near $\frac{1}{4}$ doping. *Phys. Rev. B* **85** (Jan 2012) 035414
7. Lichtenstein, J., Sánchez de la Peña, D., Rohe, D., Di Napoli, E., Honerkamp, C., Maier, S.A.: High-performance functional renormalization group calculations for interacting fermions. [arXiv:1604.06296] (April 2016)
8. Berntsen, J., Espelid, T.O., Genz, A.: Algorithm 698: DCUHRE: An adaptive multidimensional integration routine for a vector of integrals. *ACM Trans. Math. Softw.* **17**(4) (December 1991) 452–456
9. DApuzzo, M., Lapegna, M., Murli, A.: Practical aspects and experiences - Scalability and load balancing in adaptive algorithms for multidimensional integration. *Parallel Computing* **23**(8) (August 1997) 1199–1210
10. Laccetti, G., Lapegna, M.: PAMIHR. A parallel FORTRAN program for multidimensional quadrature on distributed memory architectures. *Euro-Par'99 Parallel Processing* (1999)
11. Cools, R.: Advances in multidimensional integration. *Journal of Computational and Applied Mathematics* **149**(1) (2002) 1–12
12. Krommer, A.R., Ueberhuber, C.W.: Numerical integration on advanced computer systems. Volume 848 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin (1994)
13. Trefethen, L.N.: Is Gauss Quadrature Better than Clenshaw–Curtis? *SIAM Review* **50**(1) (January 2008) 67–87
14. Berntsen, J.: Practical Error Estimation in Adaptive Multidimensional Quadrature Routines. *Journal of Computational and Applied Mathematics* **25**(3) (May 1989) 327–340