

Adaptive Multidimensional Quadrature on Multi-GPU Systems

Melanie Tonarelli^[0009-0005-1264-8159],
Simone Riva^[0000-0003-1032-0609],
Pietro Benedusi^[0000-0001-7799-5999],
Fabrizio Ferrandi^[0000-0003-0301-4419],
Rolf Krause^[0000-0001-5408-5271]

1 Introduction

The accurate and efficient evaluation of multidimensional integrals plays a fundamental role in many computational problems; for example, it is crucial in radiative transfer [1, 10, 9] and probabilistic design [7] applications. Among the available approaches, deterministic adaptive quadrature methods are particularly attractive because they provide rigorous error control through recursive refinement of the integration domain. These methods iteratively subdivide the domain into smaller subregions according to local error estimators, focussing computational effort where the integrand exhibits sharp variations or singularities. However, in distributed systems, adaptivity can introduce severe load imbalance. Subdomains where the integrand has a simple behaviour are computationally lightweight, whereas those featuring localised peaks or singularities require extensive refinement, resulting in uneven loads and limiting scalability. Efficient integration therefore demands distributed adaptivity and dynamic workload redistribution across subdomains.

Frameworks such as QUADPACK [8] and Cuba [5] implement adaptive quadrature on CPU, and PAGANI [11] extends this capability to a single GPU. We introduce

Melanie Tonarelli

Euler Institute, Faculty of Informatics, Università della Svizzera Italiana, Lugano, Switzerland & Politecnico di Milano, Milan, Italy, e-mail: melanie.tonarelli@usi.ch

Simone Riva · Pietro Benedusi

Istituto ricerche solari Aldo e Cele Daccò (IRSOL), Faculty of Informatics, Università della Svizzera Italiana, Locarno, Switzerland, e-mail: {pietro.benedusi, simone.riva}@irsol.usi.ch

Fabrizio Ferrandi

Politecnico di Milano, Milan, Italy, e-mail: fabrizio.ferrandi@polimi.it

Rolf Krause

AMCS, KAUST, King Abdullah University of Science and Technology, Thuwal, Saudi Arabia & Euler Institute, Faculty of Informatics, Università della Svizzera Italiana, Lugano, Switzerland, e-mail: rolf.krause@kaust.edu.sa

a distributed adaptive strategy that extends deterministic quadrature to multi-GPU architectures via dynamic domain decomposition and load redistribution. As shown in the experimental section, the presented strategy is efficient, robust, and accurate.

2 Adaptive quadrature methods

Given a hyper-rectangle $\Omega \subset \mathbb{R}^d$ and $f : \Omega \rightarrow \mathbb{R}$ an integrable function, a *quadrature rule* approximates an integral as a weighted sum of function values, i.e.

$$\int_{\Omega} f(\mathbf{x}) \, d\mathbf{x} \approx \sum_{i=1}^n w_i f(\mathbf{x}_i),$$

where $\{(w_i, \mathbf{x}_i)\}_{i=1}^n$ are the weights and nodes of the rule. The accuracy of a rule depends on its order and on the smoothness of the integrand. The central rule used in this work is the 9-order *Genz–Malik* (GM) [4], a fully symmetric rule (nodes are distributed preserving hypercube symmetry through coordinate permutations and sign changes), with the number of nodes scaling as $O(2^d)$, making it a practical choice for high-dimensional problems.

Adaptive quadrature methods approximate multidimensional integrals by repeatedly applying a quadrature rule to progressively refined subregions of the integration domain. At each step, every subregion yields a local integral estimate and an associated error estimate. Traditional adaptive algorithms maintain a prioritised list (e.g. a heap or stack) of subregions and, at each iteration, select the single subregion with the largest estimated error for refinement. That subregion is subdivided, and the process continues until the global error estimate falls below the desired tolerance. This strategy, also known as *h*-adaptivity, concentrates computational effort where f is highly oscillatory, peaked, or discontinuous. The effectiveness of adaptive methods strongly depends on the error estimator. We use the heuristic estimator proposed in [2], which is tailored to the GM rule, to balance accuracy with computational efficiency. Another important design choice in multidimensional refinement is how to select the axis along which to refine a subregion. As a common heuristic [5], we estimate the fourth derivative of f along each axis; we then divide the subdomain along the direction with the largest estimate, where the greatest error reduction is expected. A detailed discussion of these design choices can be found in [12].

3 Methodology

Figure 1 shows the workflow of the proposed solver. The single-GPU workflow, based on that introduced in [11], is obtained by omitting the communication steps. In contrast to traditional adaptive schemes, all subregions with a non-negligible contribution to the global error estimate are refined at each iteration. This strategy

avoids sequential bottlenecks typical of traditional methods and is therefore better suited for GPU architectures. The process begins with an initial uniform partition of the integration domain and proceeds iteratively: for each subdomain, the integrand is evaluated to obtain the local integral and error estimates, and the corresponding splitting axis. Then, global convergence is checked against a prescribed tolerance. A heuristic classifier [11] discards subregions whose contribution to the error is

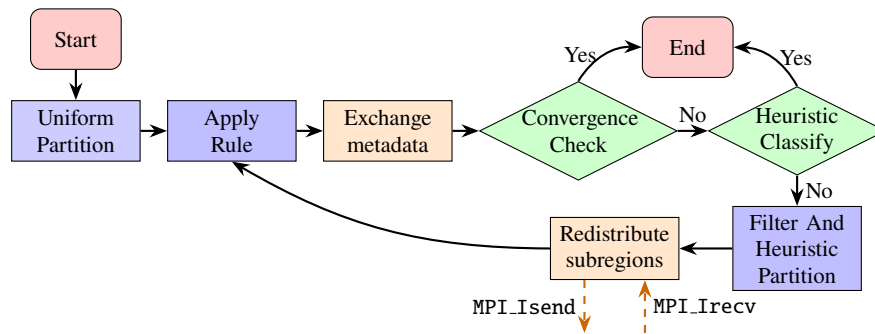


Fig. 1 Multi-GPU workflow. The single-GPU variant omits the orange communication steps and directly feeds the filtering stage into the evaluation step.

negligible, whereas the remaining ones are subdivided along their assigned axis. The resulting subregions form the input for the next iteration, and the procedure is repeated until the convergence is reached.

All subregion data remain resident on the device throughout the computation, avoiding costly transfers between host and device. Moreover, data are stored in a Structure-of-Arrays (SoA) layout, ensuring coalesced memory accesses and efficient use of GPU memory bandwidth. Building on this baseline workflow, we introduced modifications aimed at improving robustness and generality. This framework accommodates multiple quadrature rules, including a tensor-product Gauss–Kronrod rule [6] currently limited to a single GPU. We use numerical guards following [3] to mitigate round-off errors and singularities, ensuring stable convergence and preventing over-refinement. Furthermore, the filtering and splitting stages are fused into a single GPU kernel to reduce data movement and increase computational efficiency.

Extending adaptive quadrature from one GPU to many is not straightforward. A naive strategy would be to partition the integration domain uniformly across devices and let each GPU run the adaptive algorithm independently. However, depending on the behaviour of the integrand in the subdomain, some GPUs may terminate quickly while others remain overloaded. This imbalance is the main challenge of the multi-GPU extension and motivates the need for dynamic workload redistribution. We extended the single-GPU workflow to a distributed-memory setting using the Message Passing Interface (MPI). At first, the global integration domain is uniformly partitioned into more subdomains than there are devices, and these are distributed across ranks so that each GPU begins with several regions to process. This reduces the risk that one GPU is assigned only a particularly difficult region while oth-

ers finish early. Once the initial distribution is complete, each GPU executes the workflow depicted in Figure 1. Compared to the single-GPU variant, this workflow includes two additional steps: (i) *redistribution*: at the end of each iteration, the subregions are redistributed to improve load balance. Communication occurs after splitting (transferring subregion coordinates rather than full data structures) using CUDA-aware, non-blocking MPI primitives. Transfers are overlapped with GPU computation to hide latency; (ii) *exchange of metadata*: after each ‘evaluation’ step, devices exchange compact metadata summarising their local progress through MPI collectives, namely `MPI_AllReduce`. Each record includes the partial integral and error contributions, as well as “in-flight” estimates associated with subregions currently in transit, which provide conservative upper bounds on their contribution, thereby ensure correct convergence detection. This step represents the only global synchronisation point. A *redistribution policy* specifies (i) how *donors*, i.e., GPUs holding more subregions than the global fair share, are paired with *receivers*, i.e., GPUs holding fewer, and (ii) how many subregions are transferred. Transfers are constrained by a communication cap that limits the number of subregions per message. Within this bound, donors select a small batch of subregions with the largest error estimates, chosen after sorting, since these regions are most likely to require further refinement and therefore provide useful work for the receivers. Currently, the strategy relies on a *round-robin* policy, in which the devices are paired according to a cyclical schedule. The scheme is lightweight, deterministic, conflict-free and guarantees that, over successive rounds, each GPU is paired with every other GPU. Its main limitation is that when two donors or two receivers are paired, no transfer occurs in that round, which reduces the effectiveness of load balancing. Moreover, the partner sequence is not topology-aware, so some exchanges may cross nodes unnecessarily, even when the imbalance could be resolved locally.

4 Results

All experiments were carried out on the Daint Alps supercomputer at CSCS, using NVIDIA GH200 superchips with CUDA-aware MPI. We set $\Omega = [0, 1]^d$ for consistency with established benchmarks; more general hyper-rectangular domains can be handled via an affine change of variables. The stopping criterion is $\varepsilon \leq \max(10^{-16}, |I| \cdot \tau_{\text{rel}})$ with $\tau_{\text{rel}} = 10^{-k}$ (for $k = 3, \dots, 10$) a relative tolerance and ε and I the global error and integral estimates, respectively. Integrand vary in dimensions, smoothness, and difficulty, ensuring a complete stress test:

$$\begin{aligned}
 f_1(\mathbf{x}) &= \cos\left(\sum_{n=1}^d nx_n\right), & f_2(\mathbf{x}) &= \prod_{n=1}^d \frac{1}{50^{-2} + (x_n - \frac{1}{2})^2}, \\
 f_3(\mathbf{x}) &= (1 + \sum_{n=1}^d nx_n)^{-(d+n)}, & f_4(\mathbf{x}) &= \exp\left(-25^2 \sum_{n=1}^d (x_n - \frac{1}{2})^2\right), \\
 f_5(\mathbf{x}) &= \exp\left(-10 \sum_{n=1}^d |x_n - \frac{1}{2}|\right), \\
 f_6(\mathbf{x}) &= \begin{cases} 0, & x_n > (3+n)/10, n = 1, \dots, d, \\ \exp\left(\sum_{n=1}^d (n+4)x_n\right), & \text{otherwise.} \end{cases}
 \end{aligned}$$

Although these test functions are not tailored to a specific application, they capture difficult integrand behaviours encountered in practice (e.g. localised peaks, large gradients, different regularities). In radiative transfer, for example, integrands may involve combinations of complex functions (e.g., Faddeeva functions [10]) which typically exhibit localised peaks (as in f_2, \dots, f_6). Similar Gaussian-like (see f_4), localised structures also arise in probabilistic design. Highly oscillatory functions (as f_1), with jumping coefficients, (f_6) might appear in highly heterogenous problems.

On a single GPU, we evaluated our Genz–Malik (GM) solver and compared it with the state-of-the-art PAGANI framework [11]. Figure 2 (top row) reports the ex-

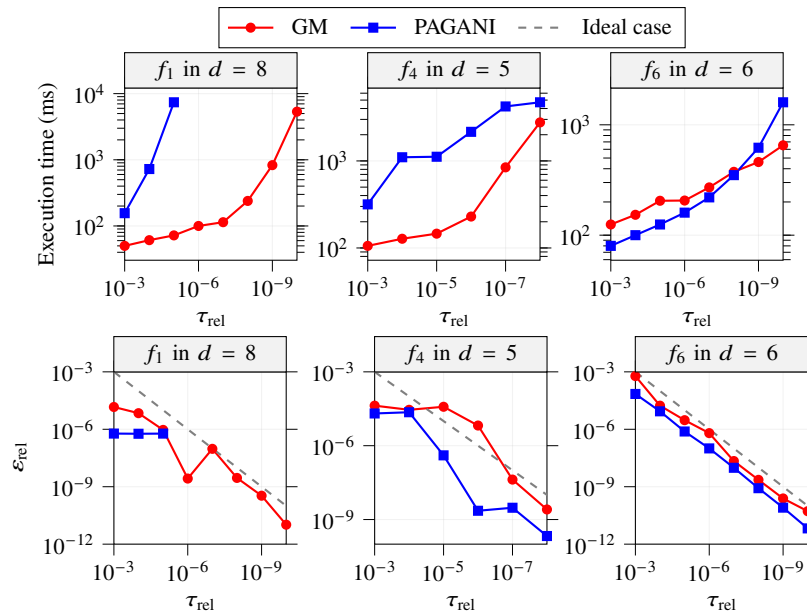


Fig. 2 Comparison between GM and PAGANI as a function of the prescribed tolerance $\tau_{\text{rel}} = 10^{-k}$ (x -axis showing k) on a single GPU. **Top row**: Execution times. **Bottom row**: Relative errors $\epsilon_{\text{rel}} = |I - I_{\text{exact}}|/|I_{\text{exact}}|$ of GM and PAGANI, with I_{exact} the exact value.

ecution times as a function of τ_{rel} for representative cases. Across these benchmarks, our strategy maintained competitive or superior runtimes. Moreover, for f_1 (highly

oscillatory), it converged for all cases, while PAGANI failed for high accuracy. For f_4 (Gaussian), our strategy was faster in all tolerance ranges. For f_6 (discontinuous), runtimes were similar for low accuracy, while our strategy became faster when increasing accuracy. In the remaining test functions (see Chapter 4 of [12]), the picture was mixed: PAGANI was generally more efficient or comparable in peaked integrands (f_2, f_3), where its aggressive pruning avoided redundant work, while our strategy proved to be more robust in discontinuous and oscillatory functions, becoming increasingly advantageous as the tolerance tightened. The accuracy results in Figure 2 (bottom row) confirm that both solvers generally met the prescribed tolerances. For f_1 , our strategy continued to refine below 10^{-6} where PAGANI stalled, while for f_4 it occasionally overshoot the target at intermediate tolerances due to overoptimistic pruning in the Gaussian tails. For all other cases, the requested accuracy was reliably achieved. For completeness, we note that the tensor-product Gauss-Kronrod variant exhibited strong accuracy and competitive performance in low to moderate dimensions, but its computational cost increased prohibitively with $d \geq 7$.

In the multi-GPU configuration, each MPI rank manages a device. At startup, the integration domain was uniformly partitioned into 8 subdomains per rank. By default, we used a message limit of 512 subregions. These values were chosen as a trade-off between initial load balance and communication overhead and affect performance and scalability but not numerical accuracy. The primary advantage of the multi-GPU extension is its ability to overcome the memory limitations of a single device. By distributing the workload, the solver is able to integrate functions at higher accuracies and dimensions, up to $d = 11$ (see Figure 3 (left)). This shows that multi-GPU execution is not only a matter of performance improvement but a prerequisite for addressing high-dimensional problems where high accuracy is required, e.g. radiative transfer problems including partial frequency redistribution [9]. Beyond feasibility,

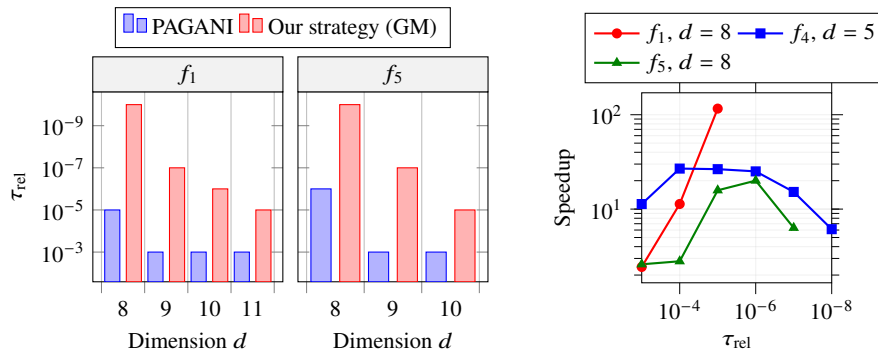


Fig. 3 Comparison between PAGANI on a single GPU and our strategy (GM) on two GPUs depending on $\tau_{\text{rel}} = 10^{-k}$. **Left:** Feasibility comparison for test functions f_1 and f_5 across different dimensions. The bars indicate the strictest relative tolerances at which convergence was achieved. **Right:** Speedup of GM w.r.t. PAGANI.

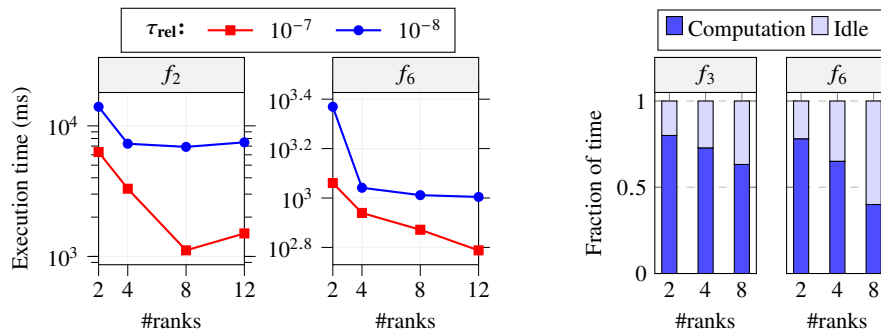


Fig. 4 Performance of the multi-GPU solver under the round-robin policy as a function of the number of MPI ranks. **Left:** Strong scaling for f_2 and f_6 in $d = 6$. **Right:** Computation and idle time fractions for f_3 and f_6 in $d = 6$ at $\tau_{rel} = 10^{-8}$.

we analysed scalability. Figure 4 (left) reports the strong scaling of the round-robin redistribution policy for representative integrands at different tolerances. Runtimes improved when moving from two to four devices, but beyond this point the benefits flattened or even reversed, with execution times increasing at 8 and 12 GPUs for several functions. This outcome is not surprising, as adaptive algorithms produce irregular workloads that hinder load balancing at scale. Three factors explain the observed behaviour: (i) *synchronisation overhead*, since each iteration requires all ranks to align at the global synch point, forcing under-loaded devices to wait for overloaded ones; (ii) *propagation of imbalance*, since the number of subregions exchanged per round is capped, heavily loaded ranks may remain overloaded across multiple iterations; (iii) *bursty and only partially overlapped communication*. Redistribution is asynchronous and overlaps partly with the ‘evaluate subregions’ phase, which dominates the iteration runtime. However, it is often still too short to cover all transfer times. Thus, many messages remain in flight when the global convergence check is reached, and pending communication accumulates at the barrier as idle time. Idle time increases rapidly with the number of GPUs, in some cases exceeding the useful compute time already at 8 devices (Fig. 4 (right)). Although suboptimal in terms of strong scaling, our solver still achieves substantial speedups over PAGANI across the tested functions. As shown in Figure 3 (right), runtimes are reduced by up to an order of magnitude at the same tolerance and dimension. This confirms that, despite limited scalability, the multi-GPU solver provides clear performance gains relative to the state-of-the-art.

5 Conclusions

We presented a general-purpose adaptive quadrature strategy for multi-GPU systems, where a decentralised redistribution policy enables highly accurate and fast numerical integration in high dimensions. Although strong scalability is limited by

synchronisation and load imbalance, the results demonstrate that multi-GPU adaptivity is essential to broaden the applicability of deterministic quadrature. Preliminary experiments with more sophisticated load balancing policies have shown potential for reducing idle time, and future work should further investigate this direction to improve the strong scaling behaviour of the current implementation. Another promising direction is the design of more informed initial domain decomposition among GPUs to alleviate the imbalance from the outset. On the application side, adaptive multi-GPU quadrature will be useful in the context of 3D radiative transfer routines,¹ enabling the efficient use of accurate scattering kernels and paving the way for high-fidelity spectropolarimetric simulations.

References

1. Benedusi, P., Riva, S., Zulian, P., Štěpán, J., Belluzzi, L., Krause, R.: Scalable matrix-free solver for 3D transfer of polarized radiation in stellar atmospheres. *J. Comput. Phys.* **479**, 112013 (2023)
2. Berntsen, J., Espelid, T.O., Genz, A.: An adaptive algorithm for the approximate calculation of multiple integrals. *ACM Trans. Math. Softw.* **17**(4), 437–451 (1991)
3. Gander, W., Gautschi, W.: Adaptive quadrature revisited. *BIT Numer. Math.* **40**, 84–101 (2000)
4. Genz, A.C., Malik, A.A.: An imbedded family of fully symmetric numerical integration rules. *SIAM J. Numer. Anal.* **20**(3), 580–588 (1983)
5. Hahn, T.: Cuba—a library for multidimensional numerical integration. *Comput. Phys. Commun.* **168**(2), 78–95 (2005)
6. Kronrod, A.S.: Nodes and weights of quadrature formulas. Consultants Bureau, New York (1965)
7. Lu, J., Darmofal, D.L.: Higher-dimensional integration with Gaussian weight for applications in probabilistic design. *SIAM J. Sci. Comput.* **26**(2), 613–624 (2004)
8. Piessens, R., de Doncker-Kapenga, E., Überhuber, C.W., Kahaner, D.K.: Quadpack: a subroutine package for automatic integration. Springer Ser. Comput. Math. (1983)
9. Riva, F., Janett, G., Belluzzi, L., del Pino Alemán, T., Alsina Ballester, E., Trujillo Bueno, J., Benedusi, P., Riva, S., Krause, R.: A numerical approach for modelling the polarisation signals of strong resonance lines with partial frequency redistribution. *Astron. Astrophys.* **699**, A233 (2025)
10. Riva, S., Guerreiro, N., Janett, G., Rossinelli, D., Benedusi, P., Krause, R., Belluzzi, L.: Assessment of the CRD approximation for the observer’s frame RIII redistribution matrix. *Astron. Astrophys.* **679**, A87 (2023)
11. Sakiotis, I., Arumugam, K., Paterno, M., Ranjan, D., Terzić, B., Zubair, M.: PAGANI: a parallel adaptive GPU algorithm for numerical integration. In: Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal., SC ’21. Assoc. Comput. Mach. (2021)
12. Tonarelli, M.: Adaptive multidimensional quadrature on multi-GPU systems. Master’s thesis, Università della Svizzera italiana, Lugano, Switzerland (2025)

¹ <https://github.com/pietrobe/TRIP>