Parallel Quasi-Monte Carlo Integration by Partitioning Low Discrepancy Sequences

Alexander Keller and Leonhard Grünschloß

Abstract A general concept for parallelizing quasi-Monte Carlo methods is introduced. By considering the distribution of computing jobs across a multiprocessor as an additional problem dimension, the straightforward application of quasi-Monte Carlo methods implies parallelization. The approach in fact partitions a single low-discrepancy sequence into multiple low-discrepancy sequences. This allows for adaptive parallel processing without synchronization, i.e. communication is required only once for the final reduction of the partial results. Independent of the number of processors, the resulting algorithms are deterministic, and generalize and improve upon previous approaches.

1 Introduction

The performance of many algorithms can be increased by parallelization and in fact parallel processors are ubiquitous. A recent survey [9, Sec. 6.4] identifies three approaches to parallel quasi-Monte Carlo integration [16]: Besides leapfrogging along a low discrepancy sequence or enumerating blocks of a low discrepancy sequence, low discrepancy sequences can be randomized. While randomization is simple, it requires a sacrifice of some convergence speed and enumerating blocks is not necessarily deterministic due to race conditions [18, Sec. 3.1]. The most desirable scheme would be deterministic for exact reproducibility and avoid any compromises on convergence.

Finance and computer graphics are among the domains that would benefit from such an approach. In the latter domain, a method for the parallel generation of pho-

Alexander Keller
NVIDIA, e-mail: keller.alexander@googlemail.com

Leonhard Grünschloß
NVIDIA / Weta Digital, e-mail: leonhard@gruenschloss.org

ton maps [1] has been introduced (see [8] for a solid introduction to photon mapping and [11] in this volume for a compact summary). Core of the method was a number theoretic argument similar to [12] that allowed for partitioning one Halton sequence into a number of sequences. Since all of these sequences were of low discrepancy, too, each job using such a subsequence consumed about a similar number of samples when independently adaptively terminated without communication for synchronization. The resulting union of samples is a complete initial segment of the Halton sequence followed by a comparatively small segment of samples, where the Halton sequence is used only partially.

As summarized in the survey [9, Sec. 6.4] and reported in [2, 12, 18, 9], transferring the approach to (t,s)-sequences in a straightforward way has defects and rank-1 lattice sequences [7] have not yet been considered.

In the following a strictly deterministic scheme is introduced: Based on a generalized and simplified argument on how to partition quadrature rules for parallel processing, efficient algorithms for generating the stream of samples inside each parallel job are derived.

2 Parallelization as an Additional Problem Dimension

The distribution of jobs over multiple processors working in parallel can be considered as one additional problem dimension.

For the example of the integration problem this means that in addition to the integrand dimensions we also integrate over the maximum number N of possibly parallel jobs. A job $j \in \{0,\dots,N-1\} \subset \mathbb{N}_0$ will be selected by the characteristic function

$$\chi_j(x') := \begin{cases} 1 & j \le x' < j+1 \\ 0 & \text{otherwise,} \end{cases}$$

that simply assigns the interval $[j, j+1)$ to the j-th job. Exploiting the fact that

$$1 = \sum_{j=0}^{N-1} \chi_j(x') \text{ for } 0 \le x' < N, \tag{1}$$

we rewrite the s-dimensional integral of a function f over the unit cube as

$$\begin{aligned} \int_{[0,1)^s} f(\mathbf{x})d\mathbf{x} &= \frac{1}{N}\int_0^N 1 \cdot \int_{[0,1)^s} f(\mathbf{x})d\mathbf{x}dx' \\ &= \sum_{j=0}^{N-1} \underbrace{\int_{[0,1)} \int_{[0,1)^s} \chi_j(N\cdot x') \cdot f(\mathbf{x})d\mathbf{x}dx'}_{=:S_j}, \end{aligned}$$

where we first added an integral over the number of jobs N, inserted the partition of one from Equation (1), and finally transformed everything to the $s+1$-dimensional unit cube.

Selecting one $s+1$-dimensional low-discrepancy sequence [16] of points $\mathbf{x}_i = (x_{i,0},\ldots,x_{i,s})$ to simultaneously approximate all summands

$$S_j \approx \frac{1}{n}\sum_{i=0}^{n-1} \chi_j(N\cdot x_{i,c})\cdot f(x_{i,0},\ldots,x_{i,c-1},x_{i,c+1},\ldots,x_{i,s}) \tag{2}$$

lends itself to a parallelization scheme: Due to the above property from Equation (1) the characteristic function[1] χ_j partitions the set of samples by job number j. In fact an arbitrarily chosen dimension c is partitioned into N equally sized intervals (see the illustration in Figure 1) and each job only consumes the points of the sequence which fall into its interval.[2]

3 Algorithms for Partitioning Low Discrepancy Sequences

Given a low discrepancy sequence $\mathbf{x}_i$, the point sets

$$P_j := \left\{\mathbf{x}_i : \chi_j(N\cdot x_{i,c}) = 1, i\in\mathbb{N}_0\right\} = \left\{\mathbf{x}_i : j \le N\cdot x_{i,c} < j+1, i\in\mathbb{N}_0\right\}$$

are low discrepancy sequences, too, because they result from a partitioning by planes perpendicular to the axis c, which does not change the order of discrepancy [16]. Similarly, any subsequence $(x_{i,0},\ldots,x_{i,c-1},x_{i,c+1},\ldots,x_{i,s})$ resulting from the omission of the component c, is of low discrepancy. In fact this can be interpreted as a simple way to partition a low discrepancy sequence into low discrepancy sequences (see the illustration in Figure 1).

For the common number theoretic constructions of quasi-Monte Carlo point sequences and a suitable choice of N, the integer part of $N\cdot x_{i,c}$ results in successive permutations of $\{0,\ldots,N-1\}$. Based on this observation we derive efficient algorithms to enumerate the set

$$P_j = \left\{\mathbf{x}_{i_{j,l}} : l\in\mathbb{N}_0\right\} \tag{3}$$

---

[1] Actually, any quadrature rule could be chosen.

[2] The partitions can also be scaled to fill the $(s+1)$-dimensional unit cube again. In other words, one could reuse the component chosen for selecting samples for each job, which is more efficient since one component less must be generated. Reformulating Equation (2) accordingly, requires only the generation of s-dimensional samples:

$$S_j \approx \frac{1}{n}\sum_{i=0}^{n-1} \chi_j(N\cdot x_{i,c})\cdot f(x_{i,1},\ldots,x_{i,c-1},N\cdot x_{i,c}-j,x_{i,c+1},\ldots,x_{i,s})$$

However, this variant is not recommended, because the resulting ensemble of samples may not be well-stratified in the dimension c.

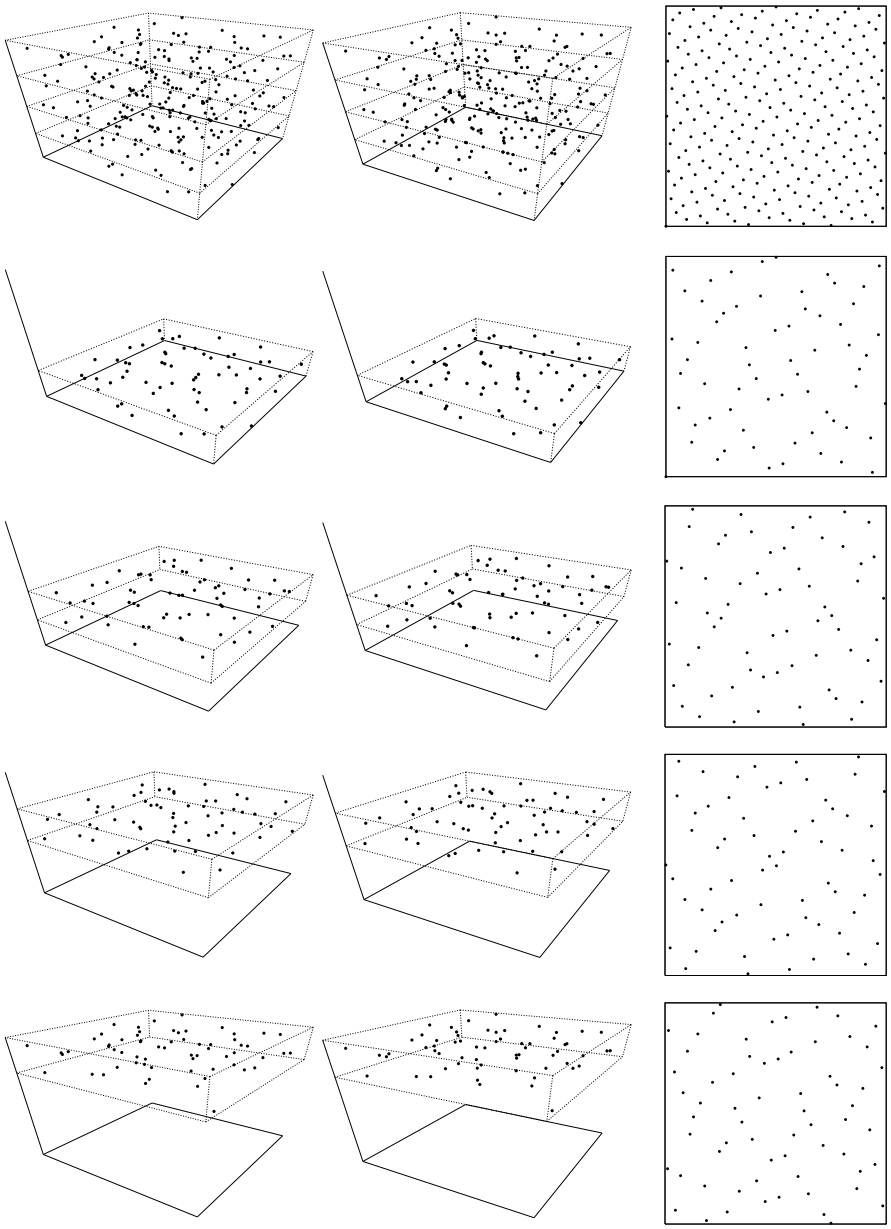

Fig. 1 The points of the three-dimensional Sobol' sequence in the first row are partitioned along the vertical axis. The resulting partitions are shown in the following four rows. Each row shows two three-dimensional plots of the same points from two perspectives that can be viewed cross-eyed, resulting in a stereoscopic impression. The rightmost plot in each row shows the two-dimensional projection of the corresponding points along the vertical axis.

for the j-th job using an index of the form $i_{j,l} := lN + k_{j,l}$, where $k_{j,l} \in \{0, \ldots, N-1\}$.

3.1 Preliminaries on Digits and Digital Radical Inverses

We introduce some notation and facts that are used throughout the derivations.

For any number $r \in \mathbb{R}_0^+$, we define the k-th digit $a_k(r) \in \{0, \ldots, b-1\}$ in integer base b by

$$r = \sum_{k=-\infty}^{\infty} a_k(r) b^k.$$

Note that this definition includes fractional digits for $k < 0$.

Digital radical inverses

$$\begin{aligned} \phi_{b,C} &: \mathbb{N}_0 \to \mathbb{Q} \cap [0,1) \\ i &\mapsto \left(b^{-1} \cdots b^{-M}\right) \left[C \begin{pmatrix} a_0(i) \\ \vdots \\ a_{M-1}(i) \end{pmatrix} \right] \end{aligned} \tag{4}$$

in base b are computed using a generator matrix C, where the inverse mapping $\phi_{b,C}^{-1}$ exists, if C is regular. While in theory these matrices are infinite-dimensional, in practice they are finite due to the finite precision of computer arithmetic. M is the number of digits, which allows for generating up to $N = b^M$ points. Note that the matrix-vector multiplications are performed in the finite field $\mathbb{F}_b$ (for the theory and mappings from and to $\mathbb{F}_b$ see [16]) and are additive in $\mathbb{F}_b$ in the sense that, for any $0 \leq M' \leq M$,

$$C \begin{pmatrix} a_0(i) \\ \vdots \\ a_{M-1}(i) \end{pmatrix} = C \begin{pmatrix} a_0(i) \\ \vdots \\ a_{M'-1}(i) \\ 0 \\ \vdots \\ 0 \end{pmatrix} + C \begin{pmatrix} 0 \\ \vdots \\ 0 \\ a_{M'}(i) \\ \vdots \\ a_{M-1}(i) \end{pmatrix}. \tag{5}$$

3.2 Halton-Type Sequences

The components

$$\begin{aligned} \phi_b &: \mathbb{N}_0 \to \mathbb{Q} \cap [0,1) \\ i = \sum_{k=0}^{\infty} a_k(i) b^k &\mapsto \sum_{k=0}^{\infty} a_k(i) b^{-k-1} \end{aligned} \tag{6}$$

of the Halton sequence [16] are the radical inverses in integer base b, where all bases are relatively prime. In fact, $\phi_b = \phi_{b,C}$ for $C = I$ the identity matrix.

While originally the digit $a_k(i)$ has been the k-th digit of the index i represented in base b, the uniformity of the points has been improved by applying permutations to the digits before computing the radical inverse: Zaremba [23] was successful with the simple permutation $\pi_b(a_k(i)) := a_k(i) + k \bmod b$, while later on Faure [5] developed a more general set of permutations improving upon Zaremba's results.

For $x_{i,c} = \phi_b(i)$ choosing the number of jobs as $N = b^m$, $m \in \mathbb{N}$, yields

$$\lfloor N \cdot x_{i,c} \rfloor = \lfloor b^m \cdot \phi_b(i) \rfloor = \left\lfloor b^m \cdot \sum_{k=0}^{\infty} a_k(i) b^{-k-1} \right\rfloor,$$

whose integer part is a permutation of $\{0,\ldots,N-1\}$, which is repeated every N points. Each job j thus is assigned the set

$$P_j = \left\{ \mathbf{x}_{l\cdot N+\phi_b^{-1}(j/N)} : l \in \mathbb{N}_0 \right\} \Leftrightarrow P_{\phi_b^{-1}(j/N)} = \left\{ \mathbf{x}_{l\cdot N+j} : l \in \mathbb{N}_0 \right\}$$

which is known as leapfrogging and coincides with previous findings in [12, 1]. Note that the offset $\phi_b^{-1}(j/N)$ is constant per job, and for the ease of implementation we can simply permute the assignment of low-discrepancy subsequences to the jobs, i.e. assign the j-th job the set

$$P_j = \left\{ \mathbf{x}_{l\cdot N+j} : l \in \mathbb{N}_0 \right\}.$$

Compared to the previous derivations [12, 1], our argument is simpler and more general as it includes scrambled radical inverses, too: The condition from [1] that N must be relatively prime to the bases of the radical inverses used for actual sampling just follows from the definition of the Halton sequence.

3.3 Digital (t,s)-Sequences in Base b

The d-th component $x_{i,d} = \phi_{b,C_d}(i)$ of a digital (t,s)-sequence in base b is computed using a generator matrix C_d, where ϕ_{b,C_d} is defined in Equation (4). Since we are considering only the algorithmic part, we refer to [16] for a more profound introduction.

The Sobol' sequence [21] is a common (t,s)-sequence in base $b = 2$. Its generator matrix for the first dimension is the identity matrix I, and thus the first component coincides with the van der Corput radical inverse in base $b = 2$ from Equation (6). As a consequence all results from the previous section apply (see the illustration in Figure 1), however, compared to the Halton sequence, the Sobol' sequence in base 2 can be computed much faster: Efficient implementations of the first two components can be found in [13] and in [22, 6] for the remaining components.

While $N = b^m$ remains a natural choice for an arbitrary C_c, determining the set P_j might not result in a simple leapfrogging scheme, because each column of the generator matrix can influence the final result.

However, if C_c is required to generate a $(0,1)$-sequence, it is known that C_c is invertible [16]. A remark in [14] states that multiplying a regular matrix to the right of all generator matrices generates the same (t,s)-sequence, except for the numbering of the points. Therefore we propose to use

$$C_0C_c^{-1},\ldots,C_{c-1}C_c^{-1},I,C_{c+1}C_c^{-1},\ldots,C_sC_c^{-1}$$

as generator matrices for the sequence. The component c then is generated by the identity matrix $I = C_cC_c^{-1}$, which allows one to efficiently determine the set P_j in analogy with the previous section.

If C_c is a regular upper triangular matrix, it is possible to compute the indices $i_{j,l}$ for every job without reordering: Due to the upper triangular structure, the m least significant digits of the index can only influence the first m digits of the radical inverse, however, the remaining index digits can influence all digits of the radical inverse, especially its first m digits. Given l and the job number j, we thus can compute

$$k_{j,l} = i_{j,l} - lb^m = \phi_{b,C_c}^{-1}\left(\sum_{k=0}^{m-1}(a_k(j) - a_{-k-1}(y_l))b^{-m+k}\right), \tag{7}$$

where the subtraction of digits has to be performed in $\mathbb{F}_b$ and $y_l = \phi_{b,C_c}(lb^m)$. This formula is best understood by first looking at the case $l = 0$, i.e. the first $N = b^m$ points, where

$$k_{j,0} = i_{j,0} = \phi_{b,C_c}^{-1}\left(\sum_{k=0}^{m-1}a_k(j)b^{-m+k}\right) = \phi_{b,C_c}^{-1}(j/N) \in \{0,\ldots,b^m-1\}$$

just is the inverse permutation generated by C_c that maps the job number to an index. For $l > 0$, the contribution of $y_l = \phi_{b,C_c}(lb^m)$ to the m least significant digits of the index has to be compensated. This is accomplished by subtracting the digits $a_{-k-1}(y_l)$ as in Equation (7) based on the additive property (5).

3.4 Rank-1 Lattice Sequences

For a suitable generator vector $\mathbf{h} \in \mathbb{N}^{s+1}$, the points of a rank-1 lattice sequence [7] are computed by

$$\mathbf{x}_i := \left\{\phi_{b,C}(i)\cdot\mathbf{h} + \Delta\right\} \in [0,1)^{s+1},$$

where $\{\cdot\}$ denotes the fractional part. The radical inverse $\phi_{b,C}$ has been defined in Equation (4) and $\Delta \in [0,1)^{s+1}$ is an arbitrary shift vector.

Restricting $C \in \mathbb{F}_b^{M\times M}$ to upper triangular, invertible matrices, the l-th run of b^m points consists of the first b^m points shifted by

$$\Delta_l := (\Delta_{l,0},\ldots,\Delta_{l,s}) = \phi_{b,C}(lb^m)\cdot\mathbf{h}.$$

We therefore choose the number of jobs to be $N = b^m$ and enumerate the points of P_j using an index $i_{j,l} := lb^m + k_{j,l}$ with $k_{j,l} \in \{0,\ldots,b^m-1\}$ of the form introduced in Equation (3). Note that under the above restriction $b^m\phi_{b,C}(k_{j,l})$ is integer.

Given j and l, $k_{j,l}$ is found by solving the following congruence resulting from taking the integer part $\lfloor\cdot\rfloor$ of the component $x_{i,c} = \{\phi_{b,C}(lb^m+k_{j,l})h_c+\Delta_c\}$ used for job selection multiplied by the number of jobs:

$$\begin{aligned}
\lfloor N\cdot x_{i,c}\rfloor &\equiv j \mod b^m\\
\lfloor b^m\{\phi_{b,C}(lb^m)h_c+\phi_{b,C}(k_{j,l})h_c+\Delta_c\}\rfloor &\equiv j \mod b^m\\
\Leftrightarrow \lfloor b^m(\phi_{b,C}(lb^m)h_c+\phi_{b,C}(k_{j,l})h_c+\Delta_c)\rfloor &\equiv j \mod b^m\\
\Leftrightarrow \lfloor b^m\phi_{b,C}(lb^m)h_c+(b^m\phi_{b,C}(k_{j,l}))h_c+b^m\Delta_c\rfloor &\equiv j \mod b^m\\
\Leftrightarrow (b^m\phi_{b,C}(k_{j,l}))h_c+\lfloor b^m\phi_{b,C}(lb^m)h_c+b^m\Delta_c\rfloor &\equiv j \mod b^m
\end{aligned}$$

and hence

$$k_{j,l} = \phi_{b,C}^{-1}\left(\underbrace{\left(\left(j-\lfloor b^m\phi_{b,C}(lb^m)h_c+b^m\Delta_c\rfloor\right)(h_c)^{-1} \bmod b^m\right)}_{\in\{0,\ldots,b^m-1\}} b^{-m}\right),$$

where $(h_c)^{-1}$ is the *modular multiplicative inverse* of h_c, which can be computed by using the extended form of Euclid's algorithm [3, Sec. 31.2, p. 937]. Note that this inverse exists if and only if b and h_c are relatively prime. For this last equation, $a \bmod b^m$ denotes the common residue, i.e. the nonnegative integer $x < b^m$, such that $a \equiv x \mod b^m$.

If now C is the identity matrix I, the inverse permutation $\phi_{b,I}^{-1} \equiv \phi_b^{-1}$ can be omitted, which is more efficient to compute and only reorders the jobs similar to Section 3.2. In addition $b^m\phi_b(lb^m) = \phi_b(l)$. An alternative and simplifying approach to the case, where the order of the points does not matter, is to multiply the generator matrix C by its inverse C^{-1} (see the aforementioned remark in [14]).

4 Parallel Quasi-Monte Carlo Integration

Considering parallelization as an additional problem dimension leads to a partition of low discrepancy sequences, where all subsequences (as described in Section 2) are of low discrepancy, too. For radical inversion based sequences, the subsequences P_j can be efficiently enumerated as derived in Section 3. In the following, more practical aspects are discussed.

4.1 Previous Issues with Leapfrogging

In literature (see the survey [9, Sec. 6.4]), leapfrogging for parallelizing quasi-Monte Carlo integration has been discussed in a controversial way. The technique has been introduced in [2], where the Sobol' low discrepancy sequence has been partitioned by leaping along the sequence in equidistant steps of size 2^m. The resulting algorithm allows for the very efficient enumeration of the subsequences, however, it "may lead to dramatic defects" [18] in parallel computation.

The underlying issue can be understood based on a simple example: The subsequences

$$\phi_b(l \cdot b^m + j) = \underbrace{b^{-m} \cdot \phi_b(l)}_{\in [0, b^{-m})} + \phi_b(j) \in [\phi_b(j), \phi_b(j) + b^{-m}) \neq [0,1) \text{ for } m > 0 \quad (8)$$

of the radical inverse (6) resulting from leapfrogging using steps of length b^m are not of low discrepancy and not even uniform over $[0,1)$ as they reside in strata [10, Sec. 3.4] not covering the whole unit interval.

In fact ϕ_2 is the first component of the Sobol' sequence, which is a (t,s)-sequence in base $b = 2$. As the first component of the subsequence identified by $j = 0$ is completely contained in $[0, 2^{-m})$, this subsequence is neither of low discrepancy nor uniform over the unit cube. In fact Lemma 8 and Remarks 9 and 10 in [19] coincide with the above explanation. As a conclusion, leaping along the sequence in equidistant steps does not guarantee that the subsequences are uniform or of low discrepancy as illustrated by the numerical experiments in [4, Fig. 3].

Obviously, these defects will not show up, if the same number of samples from all subsequences is used, but this would not allow for adaptive sampling, as discussed in the next section.

Although not aimed at parallel computation but at improving the uniformity of low discrepancy sequences, leapfrogging applied to the Halton-sequence and (t,s)-sequences has been investigated in [12], too. Similar to [1], subsequences of the Halton sequence were generated by leapfrogging with a constant step size of a prime, which is relatively prime to all bases used in the Halton sequence [12, Sec. 2.4]. While this coincides with our results, the derivation in Section 3.2 is more general, as it includes scrambled radical inverses [15] (see Section 3.3), which do not result in equidistant step size for leapfrogging.

A short investigation of the effect of leapfrogging on the Sobol' sequence in [12, Sec. 3.2] yields that for leapfrogging with step size of a power of $b = 2$, components need to be either omitted or scaled. The requirement of scaling (see also Footnote 2) can be explained with the stratification properties (8), while the omission is included in the more general results of Section 3.3.

In summary, leapfrogging works whenever one component of the sequence is used to determine the leapfrogging pattern, while the same component is excluded from sampling as discussed in Section 2.

4.2 Adaptive Sampling

In adaptive algorithms the total number of samples depends on the input and although adaptive sampling can arbitrarily fail [20], it has proved very useful in practice. Parallel adaptive sampling involves the cost of communicating termination among the processing units, load balancing to equalize for differences in performance, and potentially random elements due to race conditions.

As explained in the previous section, the defects observed in [18] will not show up with algorithms developed in Section 3, because now all subsequences are of low discrepancy. In addition, communication cost can be restricted to the minimum of the inevitable final parallel reduction operation, which sums up the partial sums of Equation (2): Running a copy of the adaptive algorithm for each subsequence, in practice each job will use about the same number of samples, because each subsequence is of low discrepancy. Using all partial sequences, the result is computed by a contiguous beginning block of samples of the underlying partitioned sequence followed by a usually quite short segment of partial samples from the underlying sequence.

Straightforward load balancing for a number P of heterogenous processors can be achieved by selecting the number of jobs $N \gg P$. Then a simple job queue can be used to distribute jobs among the processors. If the number of jobs is not excessive, the queue synchronization overhead is negligible.

The scheme is strictly deterministic, because the total number of samples of each job is independent of the processor on which the job is executed and therefore race conditions cannot occur. Computations are even independent of the number P of processors used: Executing all jobs sequentially results in exactly the same result as obtained by the parallel execution.

4.3 Selecting the Number of Jobs

Halton, (t,s)-, and lattice sequences are all based on radical inversion, which results in a number of jobs of the form $N = b^m$.

For the Halton sequence the base b is determined by the choice of the dimension c used for partitioning the sequence. Identical to [1], c and thus b should be chosen as large as possible in order to benefit from the better discrepancy and stratification properties of the first dimensions of the Halton sequence.

For (t,s)- and lattice sequences the base b is identical for all dimensions. Instead of considering which dimension to use for partitioning, it is much more interesting to choose $b = 2$, which allows for very efficient sample enumeration by using bit vector arithmetic [13, 22, 6].

Choosing the number n of samples in Equation (2) as a multiple of $N = b^m$, allows one to use finite point sets like the Hammersley points, (t,m,s)-nets, and rank-1 lattices. In this case convergence can benefit from the better discrepancy of finite point sets.

The algorithms remain consistent even for a number of jobs $N < b^m$, because each set of the partition is of low discrepancy (see Section 3). However, omitting $b^m - N$ sets of the partition is likely to sacrifice some convergence speed.

5 Conclusion

We introduced a method to partition number theoretic point sequences into subsequences that preserve the properties of the original sequence. The resulting algorithms can be classified as generalized leapfrogging [2, 18, 9]. Instead of multiplying the number of problem dimensions with the processor count [17], adding only one dimension is sufficient for our approach, which in addition allows one to benefit from lower discrepancy.

The presented algorithms are deterministic and run without races in any parallel computing environment, i.e. the computation is identical for a fixed number N of jobs no matter how many processors are used.

As a practical consequence, photon maps now can be generated adaptively in parallel similar to [1], however, taking full advantage of the much faster generation of (t,s)-sequences in base 2 [13, 22, 6], which had not been possible before.

Acknowledgements

This work has been dedicated to Stefan Heinrich's 60th birthday. The authors thank Matthias Raab for discussion.

References

1. Abramov, G.: US patent #6,911,976: System and method for rendering images using a strictly-deterministic methodology for generating a coarse sequence of sample points (2002)
2. Bromley, B.: Quasirandom number generators for parallel Monte Carlo algorithms. J. Parallel Distrib. Comput. **38**(1), 101–104 (1996)
3. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press (2009)
4. Entacher, K., Schell, T., Schmid, W., Uhl, A.: Defects in parallel Monte Carlo and quasi-Monte Carlo integration using the leap-frog technique. Parallel Algorithms Appl. pp. 13–26 (2003)
5. Faure, H.: Good permutations for extreme discrepancy. J. Number Theory **42**, 47–56 (1992)
6. Grünschloß, L.: Motion Blur. Master's thesis, Universität Ulm (2008)
7. Hickernell, F., Hong, H., L'Ecuyer, P., Lemieux, C.: Extensible lattice sequences for quasi-Monte Carlo quadrature. SIAM J. Sci. Comput. **22**, 1117–1138 (2000)
8. Jensen, H.: Realistic Image Synthesis Using Photon Mapping. AK Peters (2001)
9. Jez, P., Uhl, A., Zinterhof, P.: Applications and parallel implementation of QMC integration. In: R. Trobec, M. Vajteršic, P. Zinterhof (eds.) Parallel Computing, pp. 175–215. Springer (2008)

10. Keller, A.: Myths of computer graphics. In: H. Niederreiter (ed.) Monte Carlo and Quasi-Monte Carlo Methods 2004, pp. 217–243. Springer (2006)
11. Keller, A., Grünschloß, L., Droske, M.: Quasi-Monte Carlo progressive photon mapping. In: L. Plaskota, H. Woźniakowsi (eds.) Monte Carlo and Quasi-Monte Carlo Methods 2010, p. in this volume. Springer (2011)
12. Kocis, L., Whiten, W.: Computational investigations of low-discrepancy sequences. ACM Trans. Math. Softw. **23**(2), 266–294 (1997)
13. Kollig, T., Keller, A.: Efficient multidimensional sampling. Computer Graphics Forum (Proc. Eurographics 2002) **21**(3), 557–563 (2002)
14. Larcher, G., Pillichshammer, F.: Walsh series analysis of the L_2-discrepancy of symmetrisized point sets. Monatsh. Math. **132**, 1–18 (2001)
15. Matoušek, J.: On the L_2-discrepancy for anchored boxes. J. Complexity **14**(4), 527–556 (1998)
16. Niederreiter, H.: Random Number Generation and Quasi-Monte Carlo Methods. SIAM, Philadelphia (1992)
17. Ökten, G., Srinivasan, A.: Parallel quasi-Monte Carlo methods on a heterogeneous cluster. In: K.T. Fang, F. Hickernell, H. Niederreiter (eds.) Monte Carlo and Quasi-Monte Carlo Methods 2000, pp. 406–421. Springer (2002)
18. Schmid, W., Uhl, A.: Parallel quasi-Monte Carlo integration using (t, s)-sequences. In: ParNum '99: Proceedings of the 4th International ACPC Conference Including Special Tracks on Parallel Numerics and Parallel Computing in Image Processing, Video Processing, and Multimedia, pp. 96–106. Springer-Verlag, London, UK (1999)
19. Schmid, W., Uhl, A.: Techniques for parallel quasi-Monte Carlo integration with digital sequences and associated problems. Mathematics and Computers in Simulation **55**(1-3), 249 – 257 (2001)
20. Schwarz, H., Köckler, N.: Numerische Mathematik. 6. überarb. Auflage, Vieweg+Teubner (2008)
21. Sobol', I.: On the Distribution of points in a cube and the approximate evaluation of integrals. Zh. vychisl. Mat. mat. Fiz. **7**(4), 784–802 (1967)
22. Wächter, C.: Quasi-Monte Carlo Light Transport Simulation by Efficient Ray Tracing. Ph.D. thesis, Universität Ulm (2008)
23. Zaremba, S.: La discrépance isotrope et l'intégration numérique. Ann. Mat. Pura Appl. **87**, 125–136 (1970)