Communication Lower Bounds and Optimal Algorithms for Programs That Reference Arrays - Part 1

Michael Christ
James Demmel
Nicholas Knight
Thomas Scanlon
Katherine A. Yelick

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2013-61
http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-61.html

May 14, 2013

Acknowledgement

We acknowledge funding from Microsoft (award 024263) and Intel (award 024894), and matching funding by UC Discovery (award DIG07-10227), with additional support from ParLab affiliates National Instruments, Nokia, NVIDIA, Oracle, and Samsung, and support from MathWorks. We also acknowledge the support of the US DOE (grants DE-SC0003959, DE-SC0004938, DE-SC0005136, DE-SC0008700, DE-AC02-05CH11231, DE-FC02- 06ER25753, and DE-FC02-07ER25799), DARPA (award HR0011-12-2-0016), and NSF (grants DMS-0901569 and DMS-1001550).

Communication Lower Bounds and Optimal Algorithms for Programs That Reference Arrays — Part 1

Michael Christ[*], James Demmel[†], Nicholas Knight[‡], Thomas Scanlon[§], and Katherine Yelick[¶]

May 14, 2013

Abstract

Communication, i.e., moving data, between levels of a memory hierarchy or between parallel processors on a network, can greatly dominate the cost of computation, so algorithms that minimize communication can run much faster (and use less energy) than algorithms that do not. Motivated by this, attainable communication lower bounds were established in [12, 13, 4] for a variety of algorithms including matrix computations. The lower bound approach used initially in [13] for $\Theta(N^3)$ matrix multiplication, and later in [4] for many other linear algebra algorithms, depended on a geometric result by Loomis and Whitney [16]: this result bounded the volume of a 3D set (representing multiply-adds done in the inner loop of the algorithm) using the product of the areas of certain 2D projections of this set (representing the matrix entries available locally, i.e., without communication). Using a recent generalization of Loomis' and Whitney's result, we generalize this lower bound approach to a much larger class of algorithms, that may have arbitrary numbers of loops and arrays with arbitrary dimensions, as long as the index expressions are affine combinations of loop variables. In other words, the algorithm can do arbitrary operations on any number of variables like $A(i_1, i_2, i_2 - 2i_1, 3 - 4i_3 + 7i_4, \ldots)$. Moreover, the result applies to recursive programs, irregular iteration spaces, sparse matrices, and other data structures as long as the computation can be logically mapped to loops and indexed data structure accesses. We also discuss when optimal algorithms exist that attain the lower bounds; this leads to new asymptotically faster algorithms for several problems.

1 Introduction

Algorithms have two costs: computation (e.g., arithmetic) and communication, i.e., moving data between levels of a memory hierarchy or between processors over a network. Communication costs (measured in time or energy per operation) already greatly exceed computation costs, and the gap is growing over time following technological trends [10, 9]. Thus it is important to design algorithms that minimize communication, and if possible attain communication lower bounds. In this work, we measure communication cost in terms of the number of words moved (a bandwidth cost), and will not discuss other factors like per-message latency, congestion, or costs associated with noncontiguous data. Our goal here is to establish new lower bounds on the communication cost of a much broader class of algorithms than possible before, and when possible describe how to attain these lower bounds.

Communication lower bounds have been a subject of research for a long time. Hong and Kung [12] used an approach called *pebbling* to establish lower bounds for $\Theta(N^3)$ matrix multiplication and other algorithms. Irony, Tiskin and Toledo [13] proved the result for $\Theta(N^3)$ matrix multiplication in a different, geometric way, and extended the results both to the parallel case and the case of using redundant copies of the data. In [4] this geometric approach was further generalized to include any algorithm that "geometrically resembled" matrix multiplication in a sense to be made clear later, but included most direct linear algebra algorithms (dense or sparse, sequential or parallel), and some graph algorithms as well. Of course lower bounds alone are not algorithms, so a great deal of additional work has gone into developing algorithms that attain these lower bounds, resulting in many faster algorithms as well as remaining open problems (we discuss attainability in Section 7).

Our geometric approach, following [13, 4], works as follows. We have a set $\mathcal{Z}$ of arithmetic operations to perform, and the amount of data available locally, i.e., without any communication, is M words. For example, M could be

[*] Mathematics Dept., Univ. of California, Berkeley (mchrist@math.berkeley.edu).

[†] Computer Science Div. and Mathematics Dept., Univ. of California, Berkeley (demmel@cs.berkeley.edu).

[‡] Computer Science Div., Univ. of California, Berkeley (knight@cs.berkeley.edu).

[§] Mathematics Dept., Univ. of California, Berkeley (scanlon@math.berkeley.edu).

[¶] Computer Science Div., Univ. of California, Berkeley and Lawrence Berkeley Natl. Lab. (yelick@cs.berkeley.edu).

the cache size. Suppose we can upper bound the number of (useful) arithmetic operations that we can perform with just this data; call this bound F. Letting $|\cdot|$ denote the cardinality of a set, if the total number of arithmetic operations that we need to perform is $|\mathcal{Z}|$ (e.g., $|\mathcal{Z}| = N^3$ multiply-adds in the case of dense matrix multiplication on one processor), then we need to refill the cache at least $|\mathcal{Z}|/F$ times in order to perform all the operations. Since refilling the cache has a communication cost of moving $O(M)$ words (e.g., writing at most M words from cache back to slow memory, and reading at most M new words into cache from slow memory), the total communication cost is $\Omega(M \cdot |\mathcal{Z}|/F)$ words moved. This argument is formalized in Section 4.

The most challenging part of this argument is determining F. Our approach, described in more detail in Sections 2–3, builds on the work in [13, 4]; in those papers, the algorithm is modeled geometrically using the iteration space of the loop nest, as sketched in the following example.

Example: Matrix Multiplication (Part 1/5[1]). For N–by–N matrix multiplication, with 3 nested loops over indices (i_1, i_2, i_3), any subset E of the N^3 inner loop iterations $C(i_1, i_2) \mathrel{+}= A(i_1, i_3) \cdot B(i_3, i_2)$ can be modeled as a subset of an N–by–N–by–N discrete cube $\mathcal{Z} = \{1, \ldots, N\}^3 \subset \mathbb{Z}^3$. The data needed to execute a subset of this cube is modeled by its projections onto faces of the cube, i.e., (i_1, i_2), (i_1, i_3) and (i_3, i_2) for each $(i_1, i_2, i_3) \in E$. F is the maximum number of points whose projections are of size at most $O(M)$. In [13, 4] the authors used a geometric theorem of Loomis and Whitney (see Theorem 2.1, a special case of [16, Theorem 2]) to show F is maximized when E is a cube with edge length $M^{1/2}$, so $F = O(M^{3/2})$, yielding the communication lower bound $\Omega(M \cdot |\mathcal{Z}|/F) = \Omega(M \cdot N^3/M^{3/2}) = \Omega(N^3/M^{1/2})$. $\triangle$

Our approach is based on a major generalization of [16] in [6] that lets us geometrically model a much larger class of algorithms with an arbitrary number of loops and array expressions involving affine functions of indices.

We outline our results:

Section 2 introduces the geometric model that we use to compute the bound F (introduced above). We first describe the model for matrix multiplication (as used in [13, 4]) and apply Theorem 2.1 to obtain a bound of the form $F = O(M^{3/2})$. Then we describe how to generalize the geometric model to any program that accesses arrays inside loops with subscripts that are affine functions of the loop indices, such as $A(i_1, i_2)$, $B(i_2, i_3)$, $C(i_1 + 3i_2, 1 - i_1 + 7i_2 + 8i_3, \ldots)$, etc. (More generally, there do not need to be explicit loops or array references; see Section 4 for the general case.) In this case, we seek a bound of the form $F = O(M^{s_{\mathrm{HBL}}})$ for some constant s_{HBL}.

Section 3 takes the geometric model and in Theorem 3.2 proves a bound yielding the desired s_{HBL}. HBL stands for Hölder-Brascamp-Lieb, after the authors of precedents and generalizations of [16]. In particular, Theorem 3.2 gives the constraints for a linear program with integer coefficients whose solution is s_{HBL}.

Section 4 formalizes the argument that an upper bound on F yields a lower bound on communication of the form $\Omega(M \cdot |\mathcal{Z}|/F)$. Some technical assumptions are required for this to work. Using the same approach as in [4], we need to eliminate the possibility that an algorithm could do an unbounded amount of work on a fixed amount of data without requiring any communication. We also describe how the bound applies to communication in a variety of computer architectures (sequential, parallel, heterogeneous, etc.).

Section 5. Theorem 3.2 proves the existence of a certain set of linear constraints, but does not provide an algorithm for writing it down. Theorem 5.1 shows we can always compute a linear program with the same feasible region; i.e., the lower bounds discussed in this work are decidable. Interestingly, it may be undecidable to compute the exact set of constraints given by Theorem 3.2: Theorem 5.9 shows that having such an effective algorithm for an arbitrary set of array subscripts (that are affine functions of the loop indices) is equivalent to a positive solution to Hilbert's Tenth Problem over $\mathbb{Q}$, i.e., deciding whether a system of rational polynomial equations has a rational solution. Decidability over $\mathbb{Q}$ is a longstanding open question; this is to be contrasted with the problem over $\mathbb{Z}$, proven undecidable by Matiyasevich-Davis-Putnam-Robinson [17], or with the problem over $\mathbb{R}$, which is Tarski-decidable [21].

Section 6. While Theorem 5.1 demonstrates that our lower bounds are decidable, the algorithm given is quite inefficient. In many cases in many cases of practical interest we can write down an equivalent linear program in fewer steps; in Section 6 we present several such cases (Part 2 of this paper will discuss others). For example, when every array is subscripted by a subset of the loop indices, Theorem 6.6 shows we can obtain an equivalent linear program immediately. For example, this includes matrix multiplication, which has three loop indices i_1, i_2, and i_3, and array references $A(i_1, i_3)$, $B(i_3, i_2)$ and $C(i_1, i_2)$. Other examples include tensor contractions, the direct N-body algorithm, and database join.

[1] This indicates that this is the first of 5 times that matrix multiplication will be used as an example.

Section 7 considers when the communication lower bound for an algorithm is attainable by reordering the executions of the inner loop; we call such an algorithm *communication optimal.* For simplicity, we consider the case where all reorderings are correct. Then, for the case discussed in Theorem 6.6, i.e., when every array is subscripted by a subset of the loop indices, Theorem 7.1 shows that the dual linear program yields the "block sizes" needed for a communication-optimal sequential algorithm. Section 7.3 uses Theorem 6.6 to do the same for a parallel algorithm, providing a generalization of the "2.5D" matrix algorithms in [20]. Other examples again include tensor contractions, the direct N-body algorithm, and database join.

Section 8 summarizes our results, and outlines the contents of Part 2 of this paper. Part 2 will discuss how to compute lower bounds more efficiently and will include more cases where optimal algorithms are possible, including a discussion of loop dependencies.

This completes the outline of the paper. We note that it is possible to omit the detailed proofs in Sections 3 and 5 on a first reading; the rest of the paper is self-contained.

To conclude this introduction, we apply our theory to two examples (revisited later), and show how to derive communication-optimal sequential algorithms.

Example: Matrix Multiplication (Part 2/5). The code for computing $C = A \cdot B$ is

```
for $i_1 = 1 : N$,    for $i_2 = 1 : N$,    for $i_3 = 1 : N$,
    $C(i_1, i_2) \mathrel{+}= A(i_1, i_3) \cdot B(i_3, i_2)$
```

(We omit the details of replacing '+=' by '=' when $i_3 = 1$; this will not affect our asymptotic analysis.) We record everything we need to know about this algorithm in the following matrix Δ, which has one column for each loop index i_1, i_2, i_3, one row for each array A, B, C, and ones and zeros to indicate which arrays have which loop indices as subscripts, i.e.,

$$\Delta = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} i_1 & i_2 & i_3 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array};$$

for example, the top-left one indicates that i_1 is a subscript of A. Suppose we solve the linear program to maximize $x_1 + x_2 + x_3$ subject to $\Delta \cdot (x_1, x_2, x_3)^T \leq (1, 1, 1)^T$, getting $x_1 = x_2 = x_3 = 1/2$. Then Theorem 7.1 tells us that $s_{\mathrm{HBL}} = x_1 + x_2 + x_3 = 3/2$, yielding the well-known communication lower bound of $\Omega(N^3/M^{s_{\mathrm{HBL}}-1}) = \Omega(N^3/M^{1/2})$ for any sequential implementation with a fast memory size of M. Furthermore, a communication-optimal sequential implementation, where the only optimization permitted is reordering the execution of the inner loop iterations to enable reduced communication (e.g., using Strassen's algorithm instead is not permitted) is given by blocking the 3 loops using blocks of size M^{x_1}–by–M^{x_2}–by–M^{x_3}, i.e., $M^{1/2}$–by–$M^{1/2}$–by–$M^{1/2}$, yielding the code

```
for $j_1 = 1 : M^{x_1} : N$,    for $j_2 = 1 : M^{x_2} : N$,    for $j_3 = 1 : M^{x_3} : N$,
    for $k_1 = 0 : M^{x_1} - 1$,    for $k_2 = 0 : M^{x_2} - 1$,    for $k_3 = 0 : M^{x_3} - 1$,
        $(i_1, i_2, i_3) = (j_1, j_2, j_3) + (k_1, k_2, k_3)$
        $C(i_1, i_2) \mathrel{+}= A(i_1, i_3) \cdot B(i_3, i_2)$
```

yielding the well-known blocked algorithm where the innermost three loops multiply $M^{1/2}$–by–$M^{1/2}$ blocks of A and B and update a block of C. We note that to have all three blocks fit in fast memory simultaneously, they would have to be slightly smaller than $M^{1/2}$–by–$M^{1/2}$ by a constant factor. We will address this constant factor and others, which are important in practice, in Part 2 of this work (see Section 8). △

Example: Complicated Code (Part 1/4). The code is

```
for $i_1 = 1 : N$,    for $i_2 = 1 : N$,    for $i_3 = 1 : N$,    for $i_4 = 1 : N$,    for $i_5 = 1 : N$,    for $i_6 = 1 : N$,
    $A_1(i_1, i_3, i_6) \mathrel{+}= \mathrm{func}_1(A_2(i_1, i_2, i_4), A_3(i_2, i_3, i_5), A_4(i_3, i_4, i_6))$
    $A_5(i_2, i_6) \mathrel{+}= \mathrm{func}_2(A_6(i_1, i_4, i_5), A_3(i_3, i_4, i_6))$
```

This is not meant to be a practical algorithm but rather an example of the generality of our approach. We record everything we need to know about this program into a 6-by-6 matrix Δ with one column for every loop index $i_1, \ldots, i_6$, one row for every distinct array expression $A_1, \ldots, A_6$, and ones and zeros indicating which loop index appears as a subscript of which array. We again solve the linear program to maximize $x_1 + \cdots + x_6$ subject to $\Delta \cdot (x_1, \ldots, x_6)^T \leq (1, \ldots, 1)^T$, getting $(x_1, \ldots, x_6) = (\frac{2}{7}, \frac{3}{7}, \frac{1}{7}, \frac{2}{7}, \frac{3}{7}, \frac{4}{7})$. Then Theorem 7.1 tells us that $s_{\text{HBL}} = x_1 + \cdots + x_6 = \frac{15}{7}$, yielding the communication lower bound of $\Omega(N^6/M^{s_{\text{HBL}}-1}) = \Omega(N^6/M^{8/7})$ for any sequential implementation with a fast memory size of M. Furthermore, a communication-optimal sequential implementation is given by blocking loop index i_j by block size of M^{x_j}. In other words, the six innermost loops operate on a block of size $M^{2/7}$–by–$\cdots$–by–$M^{4/7}$. $\triangle$

Other examples appearing later include matrix-vector multiplication, tensor contractions, the direct N–body algorithm, database join, and computing matrix powers A^k.

2 Geometric Model

We begin by reviewing the geometric model of matrix multiplication introduced in [13], describe how it was extended to more general linear algebra algorithms in [4], and finally show how to generalize it to the class of programs considered in this work.

We geometrically model matrix multiplication $C = A \cdot B$ as sketched in Parts 1/5 and 2/5 of the matrix multiplication example (above). However the 'classical' (3 nested loops) algorithm is organized, we can represent it by a set of integer lattice points indexed by $\mathcal{I} = (i_1, i_2, i_3) \in \mathbb{Z}^3$. If A, B, C are N–by–N, these indices occupy an N–by–N–by–N discrete cube $\mathcal{Z} = \{1, \ldots, N\}^3$. Each point $\mathcal{I} \in \mathcal{Z}$ represents an operation $C(i_1, i_2)$+=$A(i_1, i_3) \cdot B(i_3, i_2)$ in the inner loop, and each projection of $\mathcal{I}$ onto a face of the cube represents a required operand: e.g., (i_1, i_2) represents $C(i_1, i_2)$ (see Figure 1).

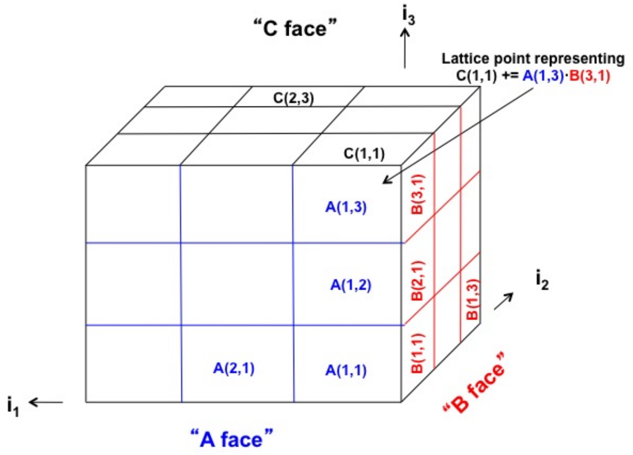

Figure 1: Modeling 3x3 Matrix Multiplication as a Lattice (each lattice point represented by a cube)

We want a bound $F \geq |E|$, where $E \subseteq \mathcal{Z}$ is any set of lattice points $\mathcal{I}$ representing operations that can be performed just using data available in fast memory of size M. Let E_A, E_B and E_C be projections of E onto faces of the cube representing all the required operands $A(i_1, i_3)$, $B(i_3, i_2)$ and $C(i_1, i_2)$, resp., needed to perform the operations represented by E. A special case of [16, Theorem 2] gives us the desired bound on E:

Theorem 2.1 (Discrete Loomis-Whitney Inequality, 3D Case). *With the above notation, for any finite subset E of the 3D integer lattice $\mathbb{Z}^3$, $|E| \leq |E_A|^{1/2} \cdot |E_B|^{1/2} \cdot |E_C|^{1/2}$.*

Finally, since the entries represented by E_A, E_B, E_C fit in fast memory by assumption (i.e., $|E_A|, |E_B|, |E_C| \leq M$), this yields the desired bound $F \geq |E|$:

$$|E| \leq |E_A|^{1/2} \cdot |E_B|^{1/2} \cdot |E_C|^{1/2} \leq M^{1/2} \cdot M^{1/2} \cdot M^{1/2} = M^{3/2} =: F \; . \tag{2.1}$$

Irony et al. [13] applied this approach to obtain a lower bound for matrix multiplication. Ballard et al. [4] extended this approach to programs of the form

$$\begin{aligned}&\text{for all } (i_1,i_2,i_3)\in\mathcal{Z}\subset\mathbb{Z}^3, \text{ in some order,}\\ &\qquad C(i_1,i_2)=C(i_1,i_2)+_{i_1,i_2}A(i_1,i_3)*_{i_1,i_2,i_3}B(i_3,i_2)\end{aligned}$$

and picked the iteration space $\mathcal{Z}$, arrays A, B, C, and binary operations $+_{i_1,i_2}$ and $*_{i_1,i_2,i_3}$ to represent many (dense or sparse) linear algebra algorithms. To make this generalization, Ballard et al. made several observations, which also apply to our case, below.

First, explicitly nested loops are not necessary, as long as one can identify each execution of the statement(s) in the inner loop with a unique lattice point; for example, many recursive computations can also match this model. Second, the upper bound F when $\mathcal{Z}$ is an N–by–N–by–N cube is valid for any subset $\mathcal{Z}\subset\mathbb{Z}^3$. Thus, we can use this bound for sparse linear algebra; sparsity may make the bound hard or impossible to attain, but it is still valid. Third, the memory locations represented by array expressions like $A(i_1,i_3)$ need not be contiguous in memory, as long as there is an injective[2] mapping from array expressions to memory locations. In other words, one can shift the index expressions like $A(i_1+3,i_3+7)$, or use pointers, or more complex kinds of data structures, to determine where the array entry actually resides in memory. Fourth, the arrays A, B and C do not have to be distinct; in the case of (in-place) LU factorization they are in fact the same, since L and U overwrite A. Rather than bounding each array, say $|E_A|\le M$, if we knew that A, B and C did not overlap, we could tighten the bound by a constant factor by using $|E_A|+|E_B|+|E_C|\le M$. (For simplicity, we will generally not worry about such constant factors in this paper, and defer their discussion to Part 2 of this work.)

Now we consider our general case, which we will refer to as the *geometric model.* This model generalizes the preceding one by allowing an arbitrary number of loops defining the index space and an arbitrary number of arrays each with arbitrary dimension. The array subscripts themselves can be arbitrary affine (integer) combinations of the loop indices, e.g., $A(i_2,i_3-2i_4,i_1+i_2+3)$.

Definition 2.2. *An instance of the geometric model is an abstract representation of an algorithm, taking the form*

$$\begin{aligned}&\text{for all } \mathcal{I}\in\mathcal{Z}\subseteq\mathbb{Z}^d, \text{ in some order,}\\ &\qquad \text{inner\_loop}(\mathcal{I},(A_1,\ldots,A_m),(\phi_1,\ldots,\phi_m))\end{aligned} \tag{2.2}$$

For $j\in\{1,\ldots,m\}$, the functions $\phi_j\colon\mathbb{Z}^d\to\mathbb{Z}^{d_j}$ are $\mathbb{Z}$–affine maps, and the functions $A_j\colon\phi_j(\mathbb{Z}^d)\to\{\text{variables}\}$ are injections into some set of variables. The subroutine inner_loop *constitutes a fairly arbitrary section of code, with the constraint that it accesses each of the variables $A_1(\phi_1(\mathcal{I})),\ldots,A_m(\phi_m(\mathcal{I}))$, either as an input, output, or both, in every iteration $\mathcal{I}\in\mathcal{Z}$.*

(We will be more concrete about what it means to 'access a variable' when we introduce our execution model in Section 4.)

Assuming that all variables are accessed in each iteration rules out certain optimizations, or requires us not to count certain inner loop iterations in our lower bound. For example, consider Boolean matrix multiplication, where the inner loop body is $C(i_1,i_2)=C(i_1,i_2)\vee(A(i_1,i_3)\wedge B(i_3,i_2))$. As long as $A(i_1,i_3)\wedge B(i_3,i_2)$ is false, $C(i_1,i_2)$ does not need to be accessed. So we need either to exclude such optimizations, or not count such loop iterations. And once $C(i_1,i_2)$ is true, its value cannot change, and no more values of $A(i_1,i_3)$ and $B(i_3,i_2)$ need to be accessed; if this optimization is performed, then these loop iterations would not occur and not be counted in the lower bound. In general, if there are data-dependent branches in the inner loop (e.g., 'flip a coin. . . '), we may not know to which subset $\mathcal{Z}$ of all the loop iterations our model applies until the code has executed. We refer to a later example (database join) and [4, Section 3.2.1] for more examples and discussion of this assumption.

Following the approach for matrix multiplication, we need to bound $|E|$ where E is a nonempty finite subset of $\mathcal{Z}$. To execute E, we need array entries $A_1(\phi_1(E)),\ldots,A_m(\phi_m(E))$. Analogous to the analysis of matrix multiplication, we need to bound $|E|$ in terms of the number of array entries needed: $|\phi_j(E)|$ entries of A_j, for $j=\{1,\ldots,m\}$.

Suppose there were nonnegative constants $s_1,\ldots,s_m$ such that for any such E,

$$|E|\le\prod_{j=1}^{m}|\phi_j(E)|^{s_j} \tag{2.3}$$

[2] We assume injectivity since otherwise an algorithm could potentially access only the same few memory locations over and over again. Given our asymptotic analysis, however, we could weaken this assumption, allowing a constant number of array entries to occupy the same memory address.

i.e., a generalization of Theorem 2.1. Then just as with matrix multiplication, if the number of entries of each array were bounded by $|\phi_j(E)| \le M$, we could bound

$$|E| \le \prod_{j=1}^{m} |\phi_j(E)|^{s_j} \le \prod_{j=1}^{m} M^{s_j} = M^{\sum_{j=1}^{m} s_j} =: M^{s_{\text{HBL}}} =: F$$

where $s_{\text{HBL}} \coloneqq \sum_{j=1}^{m} s_j$.

The next section shows how to determine the constants $s_1, \ldots, s_m$ such that inequality (2.3) holds. To give some rough intuition for the result, consider again the special case of Theorem 2.1, where $|E|$ can be interpreted as a volume, and each $|\phi_j(E)|$ as an area. Thus one can think of $|E|$ as having units of, say, meters^3, and $|\phi_j(E)|$ as having units of meters^2. Thus, for $\prod_{j=1}^{3} |\phi_j(E)|^{s_j}$ to have units of meters^3, we would expect the s_j to satisfy $3 = 2s_1 + 2s_2 + 2s_3$. But if E were a lower dimensional subset, lying say in a plane, then $|E|$ would have units meters^2 and we would get a different constraint on the s_j. This intuition is formalized in Theorem 3.2 below.

3 Upper bounds from HBL theory

As discussed above, we aim to establish an upper bound of the form (2.3) for $|E|$, for any finite set E of loop iterations. In this section we formulate and prove Theorem 3.2, which states that such a bound is valid if and only if the exponents $s_1, \ldots, s_m$ satisfy a certain finite set of linear constraints, specified in (3.1) below.

3.1 Statement and Preliminaries

As in the previous section, $\mathcal{I} = (i_1, \ldots, i_d)$ denotes a point in $\mathbb{Z}^d$. For each $j \in \{1, 2, \ldots, m\}$ we are given a $\mathbb{Z}$–affine map $\phi_j \colon \mathbb{Z}^d \to \mathbb{Z}^{d_j}$, i.e., a function that returns an affine integer combination of the coordinates $i_1, \ldots, i_d$ as a d_j–tuple, where d_j is the dimension (number of arguments in the subscript) of the array A_j. So, the action of ϕ_j can be represented as multiplication by a d_j–by–d integer matrix, followed by a translation by an integer vector of length d_j. As mentioned in Section 2, there is no loss of generality in assuming that ϕ_j is linear, rather than affine. This is because A_j can be any injection (into memory addresses), so we can always compose the translation (a bijection) with A_j. In this section we assume that this has been done already.

Notation 3.1. *We establish some group theoretic notation; for an algebra reference see, e.g., [15]. We regard the set $\mathbb{Z}^r$ (for any natural number r) as an additive Abelian group, and any $\mathbb{Z}$–linear map $\phi \colon \mathbb{Z}^r \to \mathbb{Z}^{r'}$ as a group homomorphism. An Abelian group G has a property called its* rank, *related to the dimension of a vector space: the rank of G is the cardinality of any maximal subset of G that is linearly independent. All groups discussed in this paper are Abelian and finitely generated, i.e., generated over $\mathbb{Z}$ by finite subsets. Such groups have finite ranks. For example, the rank of $\mathbb{Z}^r$ is r. The notation $H \le G$ indicates that H is a subgroup of G, and the notation $H < G$ indicates that H is a proper subgroup.*

The following theorem provides bounds which are fundamental to our application.

Theorem 3.2. *Let d and d_j be nonnegative integers and $\phi_j \colon \mathbb{Z}^d \to \mathbb{Z}^{d_j}$ be group homomorphisms for $j \in \{1, 2, \ldots, m\}$, and consider some $s \in [0,1]^m$. Suppose that*

$$\operatorname{rank}(H) \le \sum_{j=1}^{m} s_j \operatorname{rank}(\phi_j(H)) \qquad \text{for all subgroups } H \le \mathbb{Z}^d. \tag{3.1}$$

Then

$$|E| \le \prod_{j=1}^{m} |\phi_j(E)|^{s_j} \qquad \text{for all nonempty finite sets } E \subseteq \mathbb{Z}^d. \tag{3.2}$$

Conversely, if (3.2) *holds for $s \in [0,1]^m$, then s satisfies* (3.1)*.*

While (3.1) may suggest that there are an infinite number of inequalities constraining s (one for each subgroup $H \le \mathbb{Z}^d$), in fact each rank is an integer in the range $[0, d]$, and so there are at most $(d+1)^{m+1}$ different inequalities.

Note that if $s \in [0, \infty)^m$ satisfies (3.1) or (3.2), then any $t \in [0, \infty)^m$ where each $t_j \ge s_j$ does too. This is obvious for (3.1); in the case of (3.2), since E is nonempty and finite, then for each j, $1 \le |\phi_j(E)| < \infty$, so $|\phi_j(E)|^{s_j} \le |\phi_j(E)|^{t_j}$.

The following result demonstrates there is no loss of generality restricting $s \in [0,1]^m$, rather than in $[0, \infty)^m$, in Theorem 3.2.

Proposition 3.3. *Whenever* (3.1) *or* (3.2) *holds for some* $s \in [0,\infty)^m$*, it holds for* $t \in [0,1]^m$ *where* $t_j = \min(s_j, 1)$ *for* $j \in \{1,\dots,m\}$*.*

3.2 Generalizations

We formulate here two extensions of Theorem 3.2. Theorem 3.6 extends Theorem 3.2 both by replacing the sets E by functions and by replacing the codomains $\mathbb{Z}^{d_j}$ by finitely generated Abelian groups with torsion; this is a digression which is not needed for our other results. Theorem 3.10 instead replaces torsion-free finitely generated Abelian groups by finite-dimensional vector spaces over an arbitrary field. In the case this field is the set of rational numbers $\mathbb{Q}$, it represents only a modest extension of Theorem 3.2, but the vector space framework is more convenient for our method of proof. We will show in Section 3.2 how Theorem 3.10 implies Theorem 3.2 and Theorem 3.6. The main part of Section 3 will be devoted to the proof of Theorem 3.10.

Notation 3.4. *Let* X *be any set. In the discussion below,* X *will always be either a finitely generated Abelian group, or a finite-dimensional vector space over a field* $\mathbb{F}$*. For* $p \in [1,\infty)$*,* $\ell^p(X)$ *denotes the space of all* p*–power summable functions* $f\colon X \to \mathbb{C}$*, equipped with the norm*

$$\|f\|_p = \left(\sum_{x\in X} |f(x)|^p\right)^{1/p}.$$

$\ell^\infty(X)$ *is the space of all bounded functions* $f\colon X \to \mathbb{C}$*, equipped with the norm* $\|f\|_\infty = \sup_{x\in X}|f(x)|$*. These definitions apply even when* X *is uncountable: if* $p \in [1,\infty)$*, any function in* $\ell^p(X)$ *vanishes at all but countably many points in* X*, and the* ℓ^∞*–norm is still the supremum of* $|f|$*, not for instance an essential supremum defined using Lebesgue measure.*

We will use the terminology *HBL datum* to refer to either of two types of structures; the first is defined as follows:

Definition 3.5. *An* Abelian group HBL datum *is a* 3*–tuple*

$$(G,(G_j),(\phi_j)) = (G,(G_1,\dots,G_m),(\phi_1,\dots,\phi_m))$$

where G *and each* G_j *are finitely generated Abelian groups,* G *is torsion-free, each* $\phi_j\colon G \to G_j$ *is a group homomorphism, and the notation on the left-hand side always implies that* $j \in \{1,2,\dots,m\}$*.*

Theorem 3.6. *Consider an Abelian group HBL datum* $(G,(G_j),(\phi_j))$ *and* $s \in [0,1]^m$*. Suppose that*

$$\operatorname{rank}(H) \le \sum_{j=1}^m s_j \operatorname{rank}(\phi_j(H)) \qquad \textit{for all subgroups } H \le G. \tag{3.3}$$

Then

$$\sum_{x\in G}\prod_{j=1}^m f_j(\phi_j(x)) \le \prod_{j=1}^m \|f_j\|_{1/s_j} \qquad \textit{for all functions } 0 \le f_j \in \ell^{1/s_j}(G_j). \tag{3.4}$$

In particular,

$$|E| \le \prod_{j=1}^m |\phi_j(E)|^{1/s_j} \qquad \textit{for all nonempty finite sets } E \subseteq G. \tag{3.5}$$

Conversely, if (3.5) *holds for* $s \in [0,1]^m$*, then* s *satisfies* (3.8)*.*

Remark 3.7. *[6, Theorem 2.4] treats the more general situation in which* G *is not required to be torsion-free, and establishes the conclusion* (3.4) *in the weaker form*

$$\sum_{x\in G}\prod_{j=1}^m f_j(\phi_j(x)) \le C\prod_{j=1}^m \|f_j\|_{1/s_j} \qquad \textit{for all functions } 0 \le f_j \in \ell^{1/s_j}(G_j). \tag{3.6}$$

where the constant $C < \infty$ *depends only on* G *and* $\{\phi_1,\dots,\phi_m\}$*. The constant* $C = 1$ *established here is optimal. Indeed, consider any single* $x \in G$*, and for each* j *define* f_j *to be* $1_{\phi_j(x)}$*, the indicator function for* $\phi_j(x)$*. Then both sides of the inequalities in* (3.4) *are equal to* 1*. In Remark 3.11, we will show that the constant* C *is bounded above by the number of torsion elements of* G*.*

Proof of Theorem 3.2. Theorem 3.2 follows from Theorem 3.6 by setting $G = \mathbb{Z}^d$ and $G_j = \mathbb{Z}^{d_j}$ for each j. □

Proof of Theorem 3.6 (necessity). Necessity of (3.3) follows from an argument given in [6, Theorem 2.4].

Consider any subgroup $H \le G$. Let $r = \operatorname{rank}(H)$. By definition of rank, there exists a set $\{e_i\}_{i=1}^r$ of elements of H such that for any coefficients $m_i, n_i \in \mathbb{Z}$, $\sum_{i=1}^r m_i e_i = \sum_{i=1}^r n_i e_i$ only if $m_i = n_i$ for all i. For any positive integer N define E_N to be the set of all elements of the form $\sum_{i=1}^r n_i e_i$, where each n_i ranges freely over $\{1, 2, \ldots, N\}$. Then $|E_N| = N^r$.

On the other hand, for $j \in \{1, \ldots, m\}$,

$$|\phi_j(E_N)| \le A_j N^{\operatorname{rank}(\phi_j(H))}, \tag{3.7}$$

where A_j is a finite constant which depends on ϕ_j, on the structure of H_j, and on the choice of $\{e_i\}$, but not on N. Indeed, it follows from the definition of rank that for each j it is possible to permute the indices i so that for each $i > \operatorname{rank}(\phi_j(H))$ there exist integers k_i and $\kappa_{i,l}$ such that

$$k_i \phi_j(e_i) = \sum_{l=1}^{\operatorname{rank}(\phi_j(H))} \kappa_{i,l} \phi_j(e_l).$$

The upper bound (3.7) follows from these relations.

Inequality (3.2) therefore asserts that

$$N^{\operatorname{rank}(H)} \le \prod_{j=1}^m A_j^{s_j} N^{s_j \operatorname{rank}(\phi_j(H))} = A N^{\sum_{j=1}^m s_j \operatorname{rank}(\phi_j(H))}$$

where $A < \infty$ is independent of N. By letting N tend to infinity, we conclude that $\operatorname{rank}(H) \le \sum_{j=1}^m s_j \operatorname{rank}(\phi_j(H))$, as was to be shown. □

We show sufficiency in Theorem 3.6 in its full generality is a consequence of the special case in which all of the groups G_j are torsion-free.

Reduction of Theorem 3.6 (sufficiency) to the torsion-free case. Let us suppose that the Abelian group HBL datum $(G, (G_j), (\phi_j))$ and s satisfy (3.3). According to the structure theorem of finitely generated Abelian groups, each G_j is isomorphic to $\tilde{G}_j \oplus T_j$ where T_j is a finite group and $\tilde{G}_j$ is torsion-free. Here $\oplus$ denotes the direct sum of Abelian groups; $G' \oplus G''$ is the Cartesian product $G' \times G''$ equipped with the natural componentwise group law. Define $\pi_j \colon \tilde{G}_j \oplus T_j \to \tilde{G}_j$ to be the natural projection; thus $\pi_j(x, t) = x$ for $(x, t) \in \tilde{G}_j \times T_j$. Define $\tilde{\phi}_j = \pi_j \circ \phi_j \colon G \to \tilde{G}_j$.

If K is a subgroup of G_j, then $\operatorname{rank}(\pi_j(K)) = \operatorname{rank}(K)$ since the kernel T_j of π_j is a finite group. Therefore for any subgroup $H \le G$, $\operatorname{rank}(\tilde{\phi}_j(H)) = \operatorname{rank}(\phi_j(H))$. Therefore whenever $(G, (G_j), (\phi_j))$ and s satisfy the hypotheses of Theorem 3.6, $(G, (\tilde{G}_j), (\tilde{\phi}_j))$ and s also satisfy those same hypotheses.

Under these hypotheses, consider any m–tuple $f = (f_j)$ from (3.4), and for each j, define $\tilde{f}_j$ by

$$\tilde{f}_j(y) = \max_{t \in T_j} f_j(y, t), \text{ for } y \in \tilde{G}_j.$$

For any $x \in G$, $f_j(\phi_j(x)) \le \tilde{f}_j(\tilde{\phi}(x))$. Consequently $\prod_{j=1}^m f_j(\phi_j(x)) \le \prod_{j=1}^m \tilde{f}_j(\tilde{\phi}_j(x))$.

We are assuming validity of Theorem 3.6 in the torsion-free case. Its conclusion asserts that

$$\sum_{x \in G} \prod_{j=1}^m \tilde{f}_j(\tilde{\phi}_j(x)) \le \prod_{j=1}^m \|\tilde{f}_j\|_{1/s_j}.$$

For each j, $\|\tilde{f}_j\|_{1/s_j} \le \|f_j\|_{1/s_j}$. This is obvious when $s_j = 0$. If $s_j \ne 0$ then

$$\|\tilde{f}_j\|_{1/s_j}^{1/s_j} = \sum_{y \in \tilde{G}_j} \tilde{f}_j(y)^{1/s_j} = \sum_{y \in \tilde{G}_j} \max_{t \in T_j} f_j(y, t)^{1/s_j} \le \sum_{y \in \tilde{G}_j} \sum_{t \in T_j} f_j(y, t)^{1/s_j} = \|f_j\|_{1/s_j}^{1/s_j}.$$

Combining these inequalities gives the conclusion of Theorem 3.6. □

Our second generalization of Theorem 3.2 is as follows.

Notation 3.8. $\dim(V)$ *will denote the dimension of a vector space V over a field $\mathbb{F}$; all our vector spaces are finite-dimensional. Similar to our notation for subgroups, $W \le V$ indicates that W is a subspace of V, and $W < V$ indicates that W is a proper subspace.*

Definition 3.9. *A* vector space HBL datum *is a 3–tuple*

$$(V, (V_j), (\phi_j)) = (V, (V_1, \dots, V_m), (\phi_1, \dots, \phi_m))$$

where V and V_j are finite-dimensional vector spaces over a field $\mathbb{F}$, $\phi_j \colon V \to V_j$ is an $\mathbb{F}$–linear map, and the notation on the left-hand side always implies that $j \in \{1, 2, \dots, m\}$.

Theorem 3.10. *Consider a vector space HBL datum $(V, (V_j), (\phi_j))$ and $s \in [0,1]^m$. Suppose that*

$$\dim(W) \le \sum_{j=1}^{m} s_j \dim(\phi_j(W)) \qquad \text{for all subspaces } W \le V. \tag{3.8}$$

Then

$$\sum_{x \in V} \prod_{j=1}^{m} f_j(\phi_j(x)) \le \prod_{j=1}^{m} \|f_j\|_{1/s_j} \qquad \text{for all functions } 0 \le f_j \in \ell^{1/s_j}(V_j). \tag{3.9}$$

In particular,

$$|E| \le \prod_{j=1}^{m} |\phi_j(E)|^{1/s_j} \qquad \text{for all nonempty finite sets } E \subseteq V. \tag{3.10}$$

Conversely, if (3.10) *holds for $s \in [0,1]^m$, and if $\mathbb{F} = \mathbb{Q}$ or if $\mathbb{F}$ is finite, then s satisfies* (3.8)*.*

The conclusions (3.4) and (3.9) remain valid for functions f_j without the requirement that the ℓ^{1/s_j} norms are finite, under the convention that the product $0 \cdot \infty$ is interpreted as zero whenever it arises. For then if any f_j has infinite norm, then either the right-hand side is ∞ while the left-hand side is finite, or both sides are zero; in either case, the inequality holds.

Proof of Theorem 3.6 (sufficiency) in the torsion-free case. Let us suppose that the Abelian group HBL datum $(G, (G_j), (\phi_j))$ and s satisfy the hypothesis (3.3) of Theorem 3.6; furthermore suppose that each group G_j is torsion-free. G_j is isomorphic to $\mathbb{Z}^{d_j}$ where $d_j = \operatorname{rank}(G_j)$. Likewise G is isomorphic to $\mathbb{Z}^d$ where $d = \operatorname{rank}(G)$. Thus we may identify $(G, (G_j), (\phi_j))$ with $(\mathbb{Z}^d, (\mathbb{Z}^{d_j}), (\tilde{\phi}_j))$, where each homomorphism $\tilde{\phi}_j \colon \mathbb{Z}^d \to \mathbb{Z}^{d_j}$ is obtained from ϕ_j by composition with these isomorphisms. Defining scalar multiplication in the natural way (i.e., treating $\mathbb{Z}^d$ and and $\mathbb{Z}^{d_j}$ as $\mathbb{Z}$–modules), we represent each $\mathbb{Z}$–linear map $\tilde{\phi}_j$ by a matrix with integer entries.

Let $\mathbb{F} = \mathbb{Q}$. Regard $\mathbb{Z}^d$ and $\mathbb{Z}^{d_j}$ as subsets of $\mathbb{Q}^d$ and of $\mathbb{Q}^{d_j}$, respectively. Consider the vector space HBL datum $(V, (V_j), (\psi_j))$ defined as follows. Let $V = \mathbb{Q}^d$ and $V_j = \mathbb{Q}^{d_j}$, and let $\psi_j \colon \mathbb{Q}^d \to \mathbb{Q}^{d_j}$ be a $\mathbb{Q}$–linear map represented by the same integer matrix as $\tilde{\phi}_j$.

Observe that $(V, (V_j), (\psi_j))$ and s satisfy the hypothesis (3.8) of Theorem 3.10. Given any subspace $W \le V$, there exists a basis S for W over $\mathbb{Q}$ which consists of elements of $\mathbb{Z}^d$. Define $H \le \mathbb{Z}^d$ to be the subgroup generated by S (over $\mathbb{Z}$). Then $\operatorname{rank}(H) = \dim(W)$. Moreover, $\psi_j(W)$ equals the span over $\mathbb{Q}$ of $\tilde{\phi}_j(H)$, and $\dim(\psi_j(W)) = \operatorname{rank}(\tilde{\phi}_j(H))$. The hypothesis $\operatorname{rank}(H) \le \sum_{j=1}^m s_j \operatorname{rank}(\tilde{\phi}_j(H))$ is therefore equivalently written as $\dim(W) \le \sum_{j=1}^m s_j \dim(\psi_j(W))$, which is (3.8) for W.

Consider the m–tuple $f = (f_j)$ corresponding to any inequality in (3.4) for $(\mathbb{Z}^d, (\mathbb{Z}^{d_j}), (\tilde{\phi}_j))$ and s, and for each j define F_j by $F_j(y) = f_j(y)$ if $y \in \mathbb{Z}^{d_j} \subseteq \mathbb{Q}^{d_j}$, and $F_j(y) = 0$ otherwise. By Theorem 3.10,

$$\sum_{x \in \mathbb{Z}^d} \prod_{j=1}^{m} f_j(\tilde{\phi}_j(x)) = \sum_{x \in \mathbb{Q}^d} \prod_{j=1}^{m} F_j(\psi_j(x)) \le \prod_{j=1}^{m} \|F_j\|_{1/s_j} = \prod_{j=1}^{m} \|f_j\|_{1/s_j}.$$

Thus the conclusion (3.4) of Theorem 3.6 is satisfied. □

Conversely, it is possible to derive Theorem 3.10 for the case $\mathbb{F} = \mathbb{Q}$ from the special case of Theorem 3.6 in which $G = \mathbb{Z}^d$ and $G_j = \mathbb{Z}^{d_j}$ by similar reasoning involving multiplication by large integers to clear denominators.

Remark 3.11. *Suppose we relax our requirement of an Abelian group HBL datum (Definition 3.5) that the finitely generated Abelian group G is torsion-free; let us call this an HBL datum with torsion. The torsion subgroup $T(G)$ of G is the (finite) set of all elements $x \in G$ for which there exists $0 \neq n \in \mathbb{Z}$ such that $nx = 0$. As mentioned in Remark 3.7, it was shown in [6, Theorem 2.4] that for an HBL datum with torsion, the rank conditions* (3.3) *are necessary and sufficient for the existence of some finite constant C such that* (3.6) *holds. A consequence of Theorem 3.6 is a concrete upper bound for the constant C in these inequalities.*

Theorem 3.12. *Consider $(G, (G_j), (\phi_j))$, an HBL datum with torsion, and $s \in [0,1]^m$. If* (3.3) *holds, then* (3.6) *holds with $C = |T(G)|$. In particular,*

$$|E| \leq |T(G)| \cdot \prod_{j=1}^{m} |\phi_j(E)|^{s_j} \qquad \text{for all nonempty finite sets } E \subseteq G. \tag{3.11}$$

Conversely, if (3.11) *holds for $s \in [0,1]^m$, then s satisfies* (3.3).

Proof. To prove (3.6), express G as $\tilde{G} \oplus T(G)$ where $\tilde{G} \leq G$ is torsion-free. Thus arbitrary elements $x \in G$ are expressed as $x = (\tilde{x}, t)$ with $\tilde{x} \in \tilde{G}$ and $t \in T(G)$. Then $(\tilde{G}, (G_j), (\phi_j|_{\tilde{G}}))$ is an Abelian group HBL datum (with $\tilde{G}$ torsion-free) to which Theorem 3.6 can be applied. Consider any $t \in T(G)$. Define $g_j \colon G_j \to [0,\infty)$ by $g_j(x_j) = f_j(x_j + \phi_j(0,t))$. Then $f_j(\phi_j(y,t)) = f_j(\phi_j(y,0) + \phi_j(0,t)) = g_j(\phi_j(y,0))$, so

$$\sum_{y \in \tilde{G}} \prod_{j=1}^{m} f_j(\phi_j(y,t)) = \sum_{y \in \tilde{G}} \prod_{j=1}^{m} g_j(\phi_j(y,0)) \leq \prod_{j=1}^{m} \|g_j|_{\phi_j(\tilde{G})}\|_{1/s_j} \leq \prod_{j=1}^{m} \|f_j\|_{1/s_j}.$$

The first inequality is an application of Theorem 3.6. Summation with respect to $t \in T(G)$ gives the required bound.

To show necessity, we consider just the inequalities (3.11) corresponding to the subsets $E \subseteq \tilde{G}$ and follow the proof of the converse of Theorem 3.6, except substituting $A|T(G)|$ for A. □

The factor $|T(G)|$ cannot be improved if the groups G_j are torsion free, or more generally if $T(G)$ is contained in the intersection of the kernels of all the homomorphisms ϕ_j; this is seen by considering $E = T(G)$.

3.3 The polytope $\mathcal{P}$

The constraints (3.3) and (3.8) are encoded by convex polytopes in $[0,1]^m$.

Definition 3.13. *For any Abelian group HBL datum $(G, (G_j), (\phi_j))$, we denote the set of all $s \in [0,1]^m$ which satisfy* (3.3) *by $\mathcal{P}(G, (G_j), (\phi_j))$. For any vector space HBL datum $(V, (V_j), (\phi_j))$, we denote the set of all $s \in [0,1]^m$ which satisfy* (3.8) *by $\mathcal{P}(V, (V_j), (\phi_j))$.*

$\mathcal{P} = \mathcal{P}(V, (V_j), (\phi_j))$ is the subset of $\mathbb{R}^m$ defined by the inequalities $0 \leq s_j \leq 1$ for all j, and $r \leq \sum_{j=1}^{m} s_j r_j$ where $(r, r_1, \ldots, r_m)$ is any element of $\{1, 2, \ldots, \dim(V)\} \times \prod_{j=1}^{m} \{0, 1, \ldots, \dim(\phi_j(V))\}$ for which there exists a subspace $W \leq V$ which satisfies $\dim(W) = r$ and $\dim(\phi_j(W)) = r_j$ for all j. Although infinitely many candidate subspaces W must potentially be considered in any calculation of $\mathcal{P}$, there are fewer than $(\dim(V)+1)^{m+1}$ tuples $(r, r_1, \ldots, r_m)$ which generate the inequalities defining $\mathcal{P}$. Thus $\mathcal{P}$ is a convex polytope with finitely many extreme points, and is equal to the convex hull of the set of all of these extreme points. This discussion applies equally to $\mathcal{P}(G, (G_j), (\phi_j))$.

In the course of proving Theorem 3.6 above, we established the following result.

Lemma 3.14. *Let $(G, (G_j), (\phi_j))$ be Abelian group HBL datum, let $(\mathbb{Z}^d, (\mathbb{Z}^{d_j}), (\tilde{\phi}_j))$ be the associated datum with torsion-free codomains, and let $(\mathbb{Q}^d, (\mathbb{Q}^{d_j}), (\psi_j))$ be the associated vector space HBL datum. Then*

$$\mathcal{P}(G, (G_j), (\phi_j)) = \mathcal{P}(\mathbb{Z}^d, (\mathbb{Z}^{d_j}), (\tilde{\phi}_j)) = \mathcal{P}(\mathbb{Q}^d, (\mathbb{Q}^{d_j}), (\psi_j)).$$

Now we prove Proposition 3.3, i.e., that there was no loss of generality in assuming each $s_j \leq 1$.

Proof of Proposition 3.3. Proposition 3.3 concerns the case of Theorem 3.2, but in the course of its proof we will show that this result also applies to Theorems 3.6 and 3.10.

We first show the result concerning (3.1), by considering instead the set of inequalities (3.8). Suppose that a vector space HBL datum $(V, (V_j), (\phi_j))$ and $s \in [0,\infty)^m$ satisfy (3.8), and suppose that $s_k \geq 1$ for some k. Define

$t \in [0,\infty)^m$ by $t_j = s_j$ for $j \neq k$, and $t_k = 1$. Pick any subspace $W \leq V$; let $W' := W \cap \ker(\phi_k)$ and let U be a supplement of W' in W, i.e., $W = U + W'$ and $\dim(W) = \dim(U) + \dim(W')$. Since $\dim(\phi_k(U)) = \dim(U)$,

$$0 \leq \sum_{j\neq k} t_j \dim(\phi_j(U)) = (t_k - 1)\dim(U) + \sum_{j\neq k} t_j \dim(\phi_j(U)) = -\dim(U) + \sum_{j=1}^{m} t_j \dim(\phi_j(U));$$

since s satisfies (3.8) and $\dim(\phi_k(W')) = 0$, by (3.8) applied to W',

$$\dim(W') \leq \sum_{j=1}^{m} s_j \dim(\phi_j(W')) = \sum_{j=1}^{m} t_j \dim(\phi_j(W')).$$

Combining these inequalities and noting that $\phi_j(U)$ is a supplement of $\phi_j(W')$ in $\phi_j(W)$ for each j, we conclude that t also satisfies (3.8). Given an m–tuple s with multiple components $s_k > 1$, we can consider each separately and apply the same reasoning to obtain an m–tuple $t \in [0,1]^m$ with $t_j = \min(s_j, 1)$ for all j. Our desired conclusion concerning (3.1) follows from Lemma 3.14, by considering the associated vector space HBL datum $(\mathbb{Q}^d, (\mathbb{Q}^{d_j}), (\psi_j))$ and noting that the lemma was established without assuming $s_j \leq 1$. (A similar conclusion can be obtained for (3.5).)

Next, we show the result concerning (3.2). Consider the following more general situation. Let $X, X_1, \ldots, X_m$ be sets and $\phi_j \colon X \to X_j$ be functions for $j \in \{1, \ldots, m\}$. Let $s \in [0,\infty)^m$ with some $s_k \geq 1$, and suppose that $|E| \leq \prod_{j=1}^{m} |\phi_j(E)|^{s_j}$ for any finite nonempty subset $E \subseteq X$. Fix one such set E. For each $y \in \phi_k(E)$, let $E_y = \phi_k^{-1}(y) \cap E$, the preimage of y under $\phi_k\big|_E$; thus $|\phi_k(E_y)| = 1$. By assumption, $|E_y| \leq \prod_{j=1}^{m} |\phi_j(E_y)|^{s_j}$, so it follows that

$$|E_y| \leq \prod_{j\neq k} |\phi_j(E_y)|^{s_j} \leq \prod_{j\neq k} |\phi_j(E)|^{s_k} = \prod_{j\neq k} |\phi_j(E)|^{t_k}.$$

Since E can be written as the union of disjoint sets $\bigcup_{y\in\phi_k(E)} E_y$, we obtain

$$|E| = \sum_{y\in\phi_k(E)} |E_y| \leq \sum_{y\in\phi_k(E)} \prod_{j\neq k} |\phi_j(E)|^{t_k} = |\phi_k(E)| \cdot \prod_{j\neq k} |\phi_j(E)|^{t_k} = \prod_{j=1}^{m} |\phi_j(E)|^{t_k}.$$

Given an m–tuple s satisfying with multiple components $s_k > 1$, we can consider each separately and apply the same reasoning to obtain an m–tuple $t \in [0,1]^m$ with $t_j = \min(s_j, 1)$ for all j. Our desired conclusion concerning (3.2) follows by picking $(X, (X_j), (\phi_j)) := (\mathbb{Z}^d, (\mathbb{Z}^{d_j}), (\phi_j))$, and a similar conclusion can be obtained for (3.5) and (3.10).

We claim that this result can be generalized to (3.4) and (3.9), based on a comment in [6, Section 8] that it generalizes in the weaker case of (3.6). □

To prove Theorem 3.10, we show in Section 3.4 that if (3.9) holds at each extreme point s of $\mathcal{P}$, then it holds for all $s \in \mathcal{P}$. Then in Section 3.5, we show that when s is an extreme point of $\mathcal{P}$, the hypothesis (3.8) can be restated in a special form. Finally in Section 3.7 we prove Theorem 3.10 (with restated hypothesis) when s is any extreme point of $\mathcal{P}$, thus proving the theorem for all $s \in \mathcal{P}$.

3.4 Interpolation between extreme points of $\mathcal{P}$

A reference for measure theory is [11]. Let $(X_j, \mathcal{A}_j, \mu_j)$ be measure spaces for $j \in \{1, 2, \ldots, m\}$, where each μ_j is a nonnegative measure on the σ–algebra $\mathcal{A}_j$. Let S_j be the set of all simple functions $f_j \colon X_j \to \mathbb{C}$. Thus S_j is the set of all $f_j \colon X_j \to \mathbb{C}$ which can be expressed in the form $\sum_i c_i 1_{E_i}$ where $c_i \in \mathbb{C}$, $E_i \in \mathcal{A}_j$, $\mu_j(E_i) < \infty$, and the sum extends over finitely many indices i.

Let $T \colon \prod_{j=1}^{m} \mathbb{C}^{X_j} \to \mathbb{C}$ be a multilinear map; i.e., for any m–tuple $f \in \prod_{j=1}^{m} \mathbb{C}^{X_j}$ where $f_k = c_0 f_{k,0} + c_1 f_{k,1}$ for $c_0, c_1 \in \mathbb{C}$ and $f_{k,0}, f_{k,1} \in \mathbb{C}^{X_k}$,

$$T(f) = c_0 T(f_1, \ldots, f_{k-1}, f_{k,0}, f_{k+1}, \ldots, f_m) + c_1 T(f_1, \ldots, f_{k-1}, f_{k,1}, f_{k+1}, \ldots, f_m).$$

One multilinear extension of the Riesz-Thörin theorem states the following (see, e.g., [5]).

Proposition 3.15 (Multilinear Riesz–Thörin theorem). *Suppose that* $p_0 = (p_{j,0}), p_1 = (p_{j,1}) \in [1,\infty]^m$. *Suppose that there exist* $A_0, A_1 \in [0,\infty)$ *such that*

$$|T(f)| \le A_0 \prod_{j=1}^m \|f_j\|_{p_{j,0}} \qquad \text{and} \qquad |T(f)| \le A_1 \prod_{j=1}^m \|f_j\|_{p_{j,1}} \qquad \text{for all } f \in \prod_{j=1}^m S_j.$$

For each $\theta \in (0,1)$ *define exponents* $p_{j,\theta}$ *by*

$$\frac{1}{p_{j,\theta}} := \frac{\theta}{p_{j,0}} + \frac{1-\theta}{p_{j,1}}.$$

Then for each $\theta \in (0,1)$,

$$|T(f)| \le A_0^\theta A_1^{1-\theta} \prod_{j=1}^m \|f_j\|_{p_{j,\theta}} \qquad \text{for all } f \in \prod_{j=1}^m S_j.$$

Here $\|f_j\|_p = \|f_j\|_{L^p(X_j,\mathcal{A}_j,\mu_j)}$.

In the context of Theorem 3.10 with vector space HBL datum $(V,(V_j),(\phi_j))$, we consider the multilinear map

$$T(f) := \sum_{x\in V} \prod_{j=1}^m f_j(\phi_j(x))$$

representing the left-hand side in (3.9).

Lemma 3.16. *If* (3.9) *holds for every extreme point of* $\mathcal{P}$, *then it holds for every* $s \in \mathcal{P}$.

Proof. For any $\tilde{f} \in \prod_{j=1}^m S_j$, we define another m–tuple f where for each j, $f_j = |\tilde{f}_j|$ is a nonnegative simple function. By hypothesis, the inequality in (3.9) corresponding to f holds at every extreme point s of $\mathcal{P}$, giving

$$\left|T(\tilde{f})\right| \le \sum_{x\in V} \prod_{j=1}^m \left|\tilde{f}_j(\phi_j(x))\right| = \sum_{x\in V} \prod_{j=1}^m f_j(\phi_j(x)) \le \prod_{j=1}^m \|f_j\|_{1/s_j} = \prod_{j=1}^m \|\tilde{f}_j\|_{1/s_j}.$$

As a consequence of Proposition 3.15 (with constants $A_i = 1$), and the fact that any $s \in \mathcal{P}$ is a finite convex combination of the extreme points, this expression holds for any $s \in \mathcal{P}$. For any nonnegative function F_j (e.g., in $\ell^{1/s_j}(V_j)$), there is an increasing sequence of nonnegative simple functions f_j whose (pointwise) limit is F_j. Consider the m–tuple $F = (F_j)$ corresponding to any inequality in (3.9), and consider a sequence of m–tuples f which converge to F; then $\prod_{j=1}^m f_j$ also converges to $\prod_{j=1}^m F_j$. So by the monotone convergence theorem, the summations on both sides of the inequality converge as well. □

3.5 Critical subspaces and extreme points

Assume a fixed vector space HBL datum $(V,(V_j),(\phi_j))$, and let $\mathcal{P}$ continue to denote the set of all $s \in [0,1]^m$ which satisfy (3.8).

Definition 3.17. *Consider any* $s \in [0,1]^m$. *A subspace* $W \le V$ *satisfying* $\dim(W) = \sum_{j=1}^m s_j \dim(\phi_j(W))$ *is said to be a critical subspace; one satisfying* $\dim(W) \le \sum_{j=1}^m s_j \dim(\phi_j(W))$ *is said to be subcritical; and a subspace satisfying* $\dim(W) > \sum_{j=1}^m s_j \dim(\phi_j(W))$ *is said to be supercritical.* W *is said to be strictly subcritical if* $\dim(W) < \sum_{j=1}^m s_j \dim(\phi_j(W))$.

In this language, the conditions (3.8) assert that that every subspace W of V, including $\{0\}$ and V itself, is subcritical; equivalently, there are no supercritical subspaces. When more than one m–tuple s is under discussion, we sometimes say that W is critical, subcritical, supercritical or strictly subcritical with respect to s.

The goal of Section 3.5 is to establish the following:

Proposition 3.18. *Let* s *be an extreme point of* $\mathcal{P}$. *Then some subspace* $\{0\} < W < V$ *is critical with respect to* s, *or* $s \in \{0,1\}^m$.

Note that these two possibilities are not mutually exclusive.

Lemma 3.19. *If* s *is an extreme point of* $\mathcal{P}$, *and if* i *is an index for which* $s_i \notin \{0,1\}$, *then* $\dim(\phi_i(V)) \ne 0$.

Proof. Suppose $\dim(\phi_i(V)) = 0$. If $t \in [0,1]^m$ satisfies $t_j = s_j$ for all $j \neq i$, then $\sum_{j=1}^m t_j \dim(\phi_j(W)) = \sum_{j=1}^m s_j \dim(\phi_j(W))$ for all subspaces $W \leq V$, so $t \in \mathcal{P}$ as well. If $s_i \notin \{0,1\}$, then this contradicts the assumption that s is an extreme point of $\mathcal{P}$. □

Lemma 3.20. *Let s be an extreme point of $\mathcal{P}$. Suppose that no subspace $\{0\} < W \leq V$ is critical with respect to s. Then $s \in \{0,1\}^m$.*

Proof. Suppose to the contrary that for some index i, $s_i \notin \{0,1\}$. If $t \in [0,1]^m$ satisfies $t_j = s_j$ for all $j \neq i$ and if t_i is sufficiently close to s_i, then $t \in \mathcal{P}$. This again contradicts the assumption that s is an extreme point. □

Lemma 3.21. *Let s be an extreme point of $\mathcal{P}$. Suppose that there exists no subspace $\{0\} < W < V$ which is critical with respect to s. Then there exists at most one index i for which $s_i \notin \{0,1\}$.*

Proof. Suppose to the contrary that there were to exist distinct indices k, l such that neither of s_k, s_l belongs to $\{0,1\}$. By Lemma 3.19, both $\phi_k(V)$ and $\phi_l(V)$ have positive dimensions. For $\varepsilon \in \mathbb{R}$ define t by $t_j = s_j$ for all $j \notin \{k,l\}$,

$$t_k = s_k + \varepsilon \dim(\phi_l(V)) \text{ and } t_l = s_l - \varepsilon \dim(\phi_k(V)).$$

Whenever $|\varepsilon|$ is sufficiently small, $t \in [0,1]^m$. Moreover, V remains subcritical with respect to t. If $|\varepsilon|$ is sufficiently small, then every subspace $\{0\} < W < V$ remains strictly subcritical with respect to t, because the set of all parameters $(\dim(W), \dim(\phi_1(W)), \dots, \dim(\phi_m(W)))$ which arise, is finite. Thus $t \in \mathcal{P}$ for all sufficiently small $|\varepsilon|$. Therefore s is not an extreme point of $\mathcal{P}$. □

Lemma 3.22. *Let $s \in [0,1]^m$. Suppose that V is critical with respect to s. Suppose that there exists exactly one index $i \in \{1,2,\dots,m\}$ for which $s_i \notin \{0,1\}$. Then V has a subspace which is supercritical with respect to s.*

Proof. By Lemma 3.19, $\dim(\phi_i(V)) > 0$. Let K be the set of all indices k for which $s_k = 1$. The hypothesis that V is critical means that

$$\dim(V) = s_i \dim(\phi_i(V)) + \sum_{k \in K} s_k \dim(\phi_k(V)).$$

Since $s_i > 0$ and $\dim(\phi_i(V)) > 0$,

$$\sum_{k \in K} \dim(\phi_k(V)) = \sum_{k \in K} s_k \dim(\phi_k(V)) = \dim(V) - s_i \dim(\phi_i(V)) < \dim(V).$$

Consider the subspace $W \leq V$ defined by

$$W = \bigcap_{k \in K} \ker(\phi_k);$$

this intersection is interpreted to be $W = V$ if the index set K is empty. W necessarily has positive dimension. Indeed, W is the kernel of the map $\psi\colon V \to \bigoplus_{k \in K} \phi_k(V)$, defined by $\psi(x) = (\phi_k(x) : k \in K)$, where $\bigoplus$ denotes the direct sum of vector spaces. The image of ψ is isomorphic to some subspace of $\bigoplus_{k \in K} \phi_k(V)$, a vector space whose dimension $\sum_{k \in K} \dim(\phi_k(V))$ is strictly less than $\dim(V)$. Therefore $\ker(\psi) = W$ has dimension greater than or equal to $\dim(V) - \sum_{k \in K} \dim(\phi_k(V)) > 0$. Since $\phi_k(W) = \{0\}$ for all $k \in K$,

$$\begin{aligned}\sum_{j=1}^m s_j \dim(\phi_j(W)) &= s_i \dim(\phi_i(W)) + \sum_{k \in K} \dim(\phi_k(W)) \\ &= s_i \dim(\phi_i(W)) \\ &\leq s_i \dim(W).\end{aligned}$$

Since $s_i < 1$ and $\dim(W) > 0$, $s_i \dim(\phi_i(W))$ is strictly less than $\dim(W)$, whence W is supercritical. □

Proof of Proposition 3.18. Suppose that there exists no critical subspace $\{0\} < W < V$. By Lemma 3.20, either $s_j \in \{0,1\}^m$ — in which case the proof is complete — or V is critical. By Lemma 3.21, there can be at most one index i for which $s_i \notin \{0,1\}$. By Lemma 3.22, for critical V, the existence of one single such index i implies the presence of some supercritical subspace, contradicting the main hypothesis of Proposition 3.18. Thus again, $s \in \{0,1\}^m$. □

3.6 Factorization of HBL data

Notation 3.23. *Suppose V, V' are finite-dimensional vector spaces over a field $\mathbb{F}$, and $\phi\colon V \to V'$ is an $\mathbb{F}$–linear map. When considering a fixed subspace $W \leq V$, then $\phi\big|_W\colon W \to \phi(W)$ denotes the restriction of ϕ to W, also a $\mathbb{F}$–linear map. V/W denotes the quotient of V by W; elements of V/W are cosets $x + W = \{x + w : w \in W\} \subseteq V$ where $x \in V$. Thus $x + W = x' + W$ if and only if $x - x' \in W$. V/W is also a (finite-dimensional) vector space, under the definition $(x + W) + (x' + W) = (x + x') + W$; every subspace of V/W can be written as U/W for some $W \leq U \leq V$.*

Similarly we can define $V'/\phi(W)$, the quotient of V' by $\phi(W)$, and the quotient map $[\phi]\colon V/W \ni x + W \mapsto \phi(x) + \phi(W) \in V'/\phi(W)$, also a $\mathbb{F}$–linear map. For any $U/W \leq V/W$, $[\phi](U/W) = \phi(U)/\phi(W)$.

Let $(V, (V_j), (\phi_j))$ be a vector space HBL datum. To any subspace $W \leq V$ can be associated two HBL data:

$$(W, (\phi_j(W)), (\phi_j\big|_W)) \text{ and } (V/W, (V_j/\phi_j(W)), ([\phi_j]))$$

Lemma 3.24. *Given the vector space HBL datum $(V, (V_j), (\phi_j))$, for any subspace $W \leq V$,*

$$\mathcal{P}(W, (V_j), (\phi_j\big|_W)) \cap \mathcal{P}(V/W, (V_j/\phi_j(W)), ([\phi_j])) \subseteq \mathcal{P}(V, (V_j), (\phi_j)) \tag{3.12}$$

Proof. Consider any subspace $U \leq V$ and some $s \in [0,1]^m$ such that both $s \in \mathcal{P}(W, (V_j), (\phi_j\big|_W))$ and $s \in \mathcal{P}(V/W, (V_j/\phi_j(W)), ([\phi_j]))$. Then

$$\begin{aligned}
\dim(U) &= \dim((U + W)/W) + \dim(U \cap W) \\
&\leq \sum_{j=1}^m s_j \dim([\phi_j]((U + W)/W))) + \sum_{j=1}^m s_j \dim(\phi_j(U \cap W)) \\
&= \sum_{j=1}^m s_j \dim(\phi_j(U + W)/\phi_j(W)) + \sum_{j=1}^m s_j \dim(\phi_j(U \cap W)) \\
&= \sum_{j=1}^m s_j \left(\dim(\phi_j(U + W)) - \dim(\phi_j(W))\right) + \sum_{j=1}^m s_j \dim(\phi_j(U \cap W)) \\
&= \sum_{j=1}^m s_j \left(\dim(\phi_j(U) + \phi_j(W)) + \dim(\phi_j(U \cap W)) - \dim(\phi_j(W))\right) \\
&\leq \sum_{j=1}^m s_j \left(\dim(\phi_j(U) + \phi_j(W)) + \dim(\phi_j(U) \cap \phi_j(W)) - \dim(\phi_j(W))\right) \\
&= \sum_{j=1}^m s_j \dim(\phi_j(U)).
\end{aligned}$$

The last inequality is a consequence of the inclusions $\phi_j(U \cap W) \subseteq \phi_j(U) \cap \phi_j(W)$. The last equality is the relation $\dim(A) + \dim(B) = \dim(A + B) + \dim(A \cap B)$, which holds for any subspaces A, B of a vector space. Thus U is subcritical. □

Lemma 3.25. *Given the vector space HBL datum $(V, (V_j), (\phi_j))$, let $s \in \mathcal{P}(V, (V_j), (\phi_j))$. Let $W \leq V$ be a subspace which is critical with respect to s. Then*

$$\mathcal{P}(V, (V_j), (\phi_j)) = \mathcal{P}(W, (V_j), (\phi_j\big|_W)) \cap \mathcal{P}(V/W, (V_j/\phi_j(W)), ([\phi_j])). \tag{3.13}$$

Proof. With Lemma 3.24 in hand, it remains to show that $\mathcal{P}(V, (V_j), (\phi_j))$ is contained in the intersection of the other two polytopes.

Any subspace $U \leq W$ is also a subspace of V. U is subcritical with respect to s when regarded as a subspace of W, if and only if U is subcritical when regarded as a subspace of V. So $s \in \mathcal{P}(W, (V_j), (\phi_j\big|_W))$.

Now consider any subspace of $U/W \leq W/V$; we have $W \leq U \leq V$ and $\dim(U/W) = \dim(U) - \dim(W)$. Moreover,

$$\dim([\phi_j](U/W)) = \dim(\phi_j(U)/\phi_j(W)) = \dim(\phi_j(U)) - \dim(\phi_j(W)).$$

Therefore since $\dim(W) = \sum_{j=1}^m s_j \dim(\phi_j(W))$,

$$\begin{aligned}\dim(U/W) = \dim(U) - \dim(W) &\le \sum_{j=1}^m s_j \dim(\phi_j(U^*)) - \sum_{j=1}^m s_j \dim(\phi_j(W)) \\ &= \sum_{j=1}^m s_j \left(\dim(\phi_j(U)) - \dim(\phi_j(W))\right) = \sum_{j=1}^m s_j \dim([\phi_j](U/W))\end{aligned}$$

by the subcriticality of U, which holds because $s \in \mathcal{P}(V, (V_j), (\phi_j))$. Thus any $U/W \le V/W$ is subcritical with respect to s, so $s \in \mathcal{P}(V/W, (V_j/\phi_j(W)), ([\phi_j]))$ as well. □

3.7 Proof of Theorem 3.10

Recall we are given the vector space HBL datum $(V, (V_j), (\phi_j))$; we prove Theorem 3.10 by induction on the dimension of the ambient vector space V. If $\dim(V) = 0$ then V has a single element, and the result (3.9) is trivial.

To establish the inductive step, consider any extreme point s of $\mathcal{P}$. According to Proposition 3.18, there are two cases which must be analyzed. We begin with the case in which there exists a critical subspace $\{0\} < W < V$, which we prove in the following lemma. We assume that Theorem 3.10 holds for all HBL data for which the ambient vector space has strictly smaller dimension than is the case for the given datum.

Lemma 3.26. *Let $(V, (V_j), (\phi_j))$ be a vector space HBL datum, and let $s \in \mathcal{P}(V, (V_j), (\phi_j))$. Suppose that subspace $\{0\} < W < V$ is critical with respect to s. Then* (3.9) *holds for this s.*

Proof. Consider any inequality in (3.9). We may assume that none of the exponents equal zero. For if $s_k = 0$, then $f_k(\phi_k(x)) \le \|f_k\|_{1/s_k}$ for all x, and therefore

$$\sum_{x \in V} \prod_{j=1}^m f_j(\phi_j(x)) \le \|f_k\|_{1/s_k} \cdot \sum_{x \in V} \prod_{j \ne k} f_j(\phi_j(x)).$$

If $\|f_k\|_{1/s_k} = 0$, then (3.9) holds with both sides 0. Otherwise we divide by $\|f_k\|_{1/s_k}$ to conclude that $s \in \mathcal{P}(V, (V_j), (\phi_j))$ if and only if $(s_j)_{j \ne k}$ belongs to the polytope associated to the HBL datum $(V, (V_j)_{j \ne k}, (\phi_j)_{j \ne k})$. Thus the index k can be eliminated. This reduction can be repeated to remove all indices which equal zero.

Let $W_j := \phi_j(W)$. By Lemma 3.25, $s \in \mathcal{P}(W, (W_j), (\phi_j|_W))$. Therefore by the inductive hypothesis, one of the inequalities in (3.9) is

$$\sum_{x \in W} \prod_{j=1}^m f_j(\phi_j(x)) \le \prod_{j=1}^m \|f_j|_{W_j}\|_{1/s_j}. \tag{3.14}$$

Define $F_j \colon V_j/W_j \to [0, \infty)$ to be the function

$$F_j(x + W_j) = \Big(\sum_{y \in W_j} f_j(y + x)^{1/s_j}\Big)^{s_j}.$$

This quantity is a function of the coset $x + W_j$ alone, rather than of x itself, because for any $z \in W_j$,

$$\sum_{y \in W_j} f_j(y + (x + z))^{1/s_j} = \sum_{y \in W_j} f_j(y + x)^{1/s_j}$$

by virtue of the substitution $y + z \mapsto y$. Moreover,

$$\|F_j\|_{1/s_j} = \|f_j\|_{1/s_j}. \tag{3.15}$$

To prove this, choose one element $x \in V_j$ for each coset $x + W_j \in V_j/W_j$. Denoting by X the set of all these representatives,

$$\|F_j\|_{1/s_j}^{1/s_j} = \sum_{x \in X} \sum_{y \in W_j} f_j(y + x)^{1/s_j} = \sum_{z \in V_j} f_j(z)^{1/s_j}$$

because the map $X \times W_j \ni (x, y) \mapsto x + y \in V_j$ is a bijection.

The inductive bound (3.14) can be equivalently written in the more general form

$$\sum_{x \in W} \prod_{j=1}^{m} f_j(\phi_j(x+y)) \le \prod_{j=1}^{m} F_j([\phi_j](y+W)) \tag{3.16}$$

for any $y \in V$, by applying (3.14) to $(\hat{f}_j)$ where $\hat{f}_j(z) = f_j(z + \phi_j(y))$.

Denote by $Y \subseteq V$ a set of representatives of the cosets $y + W \in V/W$, and identify V/W with Y. Then

$$\sum_{x \in V} \prod_{j=1}^{m} f_j(\phi_j(x)) = \sum_{y \in Y} \sum_{x \in W} \prod_{j=1}^{m} f_j(\phi_j(y+x)) \le \sum_{y+W \in V/W} \prod_{j=1}^{m} F_j([\phi_j](y+W))$$

by (3.16).

By (3.15), it suffices to show that

$$\sum_{y+W \in V/W} \prod_{j} F_j([\phi_j](y+W)) \le \prod_{j} \|F_j\|_{1/s_j} \qquad \text{for all functions } 0 \le F_j \in \ell^{1/s_j}(V_j/W_j). \tag{3.17}$$

This is a set of inequalities of exactly the form (3.9), with $(V, (V_j), (\phi_j))$ replaced by $(V/W, (V_j/W_j), ([\phi_j]))$. By Lemma 3.25, $s \in \mathcal{P}(V/W, (V_j/W_j), ([\phi_j]))$, and since $\dim(V/W) < \dim(V)$, we conclude directly from the inductive hypothesis that (3.17) holds, concluding the proof of Lemma 3.26. □

According to Proposition 3.18, in order to complete the proof of Theorem 3.10, it remains only to analyze the case where the extreme point $s \in \{0,1\}^m$. Let $K = \{k : s_k = 1\}$. Consider $W = \bigcap_{k \in K} \ker(\phi_k)$. Since W is subcritical by hypothesis,

$$\dim(W) \le \sum_{j=1}^{m} s_j \dim(\phi_j(W)) = \sum_{k \in K} \dim(\phi_k(W)) = 0$$

so $\dim(W) = 0$, that is, $W = \{0\}$. Therefore the map $x \mapsto (\phi_k(x))_{k \in K}$ from V to the Cartesian product $\prod_{k \in K} V_k$ is injective.

For any $x \in V$,

$$\prod_{j=1}^{m} f_j(\phi_j(x)) \le \prod_{k \in K} f_k(\phi_k(x)) \prod_{i \notin K} \|f_i\|_\infty = \prod_{k \in K} f_k(\phi_k(x)) \prod_{i \notin K} \|f_i\|_{1/s_i}$$

since $s_i = 0$ for all $i \notin K$. Thus it suffices to prove that

$$\sum_{x \in V} \prod_{k \in K} f_k(\phi_k(x)) \le \prod_{k \in K} \|f_k\|_1.$$

This is a special case of the following result.

Lemma 3.27. *Let V be any finite-dimensional vector space over $\mathbb{F}$. Let K be a finite index set, and for each $k \in K$, let ϕ_k be an $\mathbb{F}$–linear map from V to a finite-dimensional vector space V_k. If $\bigcap_{k \in K} \ker(\phi_k) = \{0\}$ then for all functions $f_k \colon V_k \to [0, \infty)$,*

$$\sum_{x \in V} \prod_{k \in K} f_k(\phi_k(x)) \le \prod_{k \in K} \|f_k\|_1.$$

Proof. Define $\Phi \colon V \to \prod_{k \in K} V_k$ by $\Phi(x) = (\phi_k(x))_{k \in K}$. The hypothesis $\bigcap_{k \in K} \ker(\phi_k) = \{0\}$ is equivalent to Φ being injective. The product $\prod_{k \in K} \|f_k\|_1$ can be expanded as the sum of products

$$\sum_{y} \prod_{k \in K} f_k(y_k)$$

where the sum is taken over all $y = (y_k)_{k \in K}$ belonging to the Cartesian product $\prod_{k \in K} V_k$. The quantity of interest,

$$\sum_{x \in V} \prod_{k \in K} f_k(\phi_k(x)),$$

is likewise a sum of such products. Each term of the latter sum appears as a term of the former sum, and by virtue of the injectivity of Φ, appears only once. Since all summands are nonnegative, the former sum is greater than or equal to the latter. Therefore

$$\prod_{k\in K} \|f_k\|_1 = \sum_y \prod_{k\in K} f_k(y_k) \geq \sum_{x\in V} \prod_{k\in K} f_k(\phi_k(x)).$$

□

Having shown sufficiency for extreme points of $\mathcal{P}$, we apply Lemma 3.16 to conclude sufficiency for all $s \in \mathcal{P}$.

As mentioned above, necessity in the case $\mathbb{F} = \mathbb{Q}$ can be deduced from necessity in Theorem 3.2, by clearing denominators. First, we identify V and V_j with $\mathbb{Q}^d$ and $\mathbb{Q}^{d_j}$ and let E be any nonempty finite subset of $\mathbb{Q}^d$. Let $\hat{\phi}_j \colon \mathbb{Q}^d \to \mathbb{Q}^{d_j}$ be the linear map represented by the matrix of ϕ_j multiplied by the lowest common denominator of its entries, i.e., an integer matrix. Likewise, let $\hat{E}$ be the set obtained from E by multiplying each point by the lowest common denominator of the coordinates of all points in E. Then by linearity,

$$|\hat{E}| = |E| \leq \prod_{j=1}^m |\phi_j(E)|^{s_j} = \prod_{j=1}^m |\hat{\phi}_j(\hat{E})|^{s_j}.$$

Recognizing $(\mathbb{Z}^d, (\mathbb{Z}^{d_j}), (\hat{\phi}_j|_{\mathbb{Z}^d}))$ as an Abelian group HBL datum, we conclude (3.1) for this datum from the converse of Theorem 3.2. According to Lemma 3.14, (3.8) holds for the vector space HBL datum $(\mathbb{Q}^d, (\mathbb{Q}^{d_j}), (\hat{\phi}_j))$; our conclusion follows since $\dim(\hat{\phi}_j(W)) = \dim(\phi_j(W))$ for any $W \leq \mathbb{Q}^d$.

It remains to treat the case of a finite field $\mathbb{F}$. Whereas the above reasoning required only the validity of (3.10) in the weakened form $|E| \leq C \prod_{j=1}^m |\phi_j(E)|^{s_j}$ for some constant $C < \infty$ independent of E (see proof of necessity for Theorem 3.6), now the assumption that this holds with $C = 1$ becomes essential. Let W be any subspace of V. Since $|\mathbb{F}| < \infty$ and W has finite dimension over $\mathbb{F}$, W is a finite set and the hypothesis (3.10) can be applied with $E = W$. Therefore $|W| \leq \prod_{j=1}^m |\phi_j(W)|^{s_j}$. This is equivalent to

$$|\mathbb{F}|^{\dim(W)} \leq \prod_{j=1}^m |\mathbb{F}|^{s_j \dim(\phi_j(W))},$$

so since $|\mathbb{F}| \geq 2$, taking base–$|\mathbb{F}|$ logarithms of both sides, we obtain $\dim(W) \leq \sum_{j=1}^m s_j \dim(\phi_j(W))$, as was to be shown. □

4 Communication lower bounds from Theorem 3.2

In this section we introduce a concrete execution model for programs running on a sequential machine with a two-level memory hierarchy. With slight modification, our model also applies to parallel executions; we give details below in order to extend our sequential lower bounds to the parallel case in Section 4.6. We assume the memory hierarchy is program-managed, so all data movement between slow and fast memory is seen as explicit instructions, and computation can only be performed on values in fast memory. We combine this with the geometric model from Section 2 and the upper bound from Theorem 3.2 to get a communication lower bound of the form $\Omega(|\mathcal{Z}|/M^{s_{\text{HBL}}-1})$, where $|\mathcal{Z}|$ is the number of inner loop iterations, represented by the finite set $\mathcal{Z} \subseteq \mathbb{Z}^d$. Then we discuss how the lower bounds extend to programs on more complicated machines, like heterogeneous parallel systems.

In addition to the concrete execution model and the geometric model, we use pseudocode in our examples. At a high level, these three different algorithm representations are related as follows:

- A concrete execution is a sequence of instructions executed by the machine, according to the model detailed in Section 4.1. The concrete execution, unlike either the geometric model or pseudocode, contains explicit data movement operations between slow and fast memory.

- The geometric model (Definition 2.2), is the most abstract, and is the foundation of the bounds in Section 3. Each instance (2.2) of the geometric model corresponds to a set of concrete executions, as detailed in Section 4.2.

- A pseudocode representation, like our examples with (nested) for-loops, identifies an instance (2.2) of the geometric model. Since the bounds in Section 3 only depend on $(\mathbb{Z}^d, (\mathbb{Z}^{d_j}), (\phi_j))$, one may vary the iteration space $\mathcal{Z} \subseteq \mathbb{Z}^d$, the order it is traversed, and the statements in the inner loop body (provided all m arrays are still accessed each iteration), to obtain a different instance (2.2) of the geometric model with the same bound. So, when we prove a bound for a program given as pseudocode, we are in fact proving a bound for a larger class of programs.

The rest of this section is organized as follows. Section 4.1 describes the concrete execution model mentioned above, and Section 4.2 relates the concrete execution model to the geometric model. Section 4.3 states and proves the main communication lower bound result of this paper, Theorem 4.1. Section 4.4 presents a number of examples showing why the assumptions of Theorem 4.1 are in fact necessary to obtain a lower bound. Section 4.5 looks at one of these assumptions in more detail ("no loop splitting"), and shows that loop splitting can only improve (reduce) the lower bound. Finally, Section 4.6 discusses generalizations of the lower bound result to other machine models.

4.1 Concrete execution model

The hypothetical machine in our execution model has a two-level memory hierarchy: a slow memory of unbounded capacity and a fast memory that can store M words (all data in our model have one-word width). Data movement between slow and fast memory is explicitly managed by software instructions (unlike a hardware cache), and data is copied[3] at a one-word granularity (we will discuss spatial locality in Part 2 of this work). Every storage location in slow and fast memory (including the array elements $A_j(\phi_j(\mathcal{I}))$) has a unique memory address, called a *variable*; since the slow and fast memory address spaces (variable sets) are disjoint, we will distinguish between slow memory variables and fast memory variables. When a fast memory variable represents a copy of a slow memory variable v, we refer to v as a *cached slow memory variable*; in this case, we assume we can always identify the corresponding fast memory variable given v, even if the copy is relocated to another fast memory location.

We define a *sequential execution* $E = (e_1, e_2, \ldots, e_n)$ as a sequence of *statements* e_i of the following types:

Read $read(v)$: allocates a location in fast memory and copies variable v from slow to fast memory.

Write $write(v)$: copies variable v from fast to slow memory and deallocates the location in fast memory.

Compute $compute(\{v_1, \ldots, v_n\})$ is a statement accessing variables $v_1, \ldots, v_n$.

Allocate $allocate(v)$ introduces variable v in fast memory.

Free $free(v)$ removes variable v from fast memory.

A sequential execution defines a total order on the statements, and thereby a natural notion of when one statement succeeds or precedes another. We say that a Read or Allocate statement and a subsequent Write or Free statement are *paired* if the same variable v appears as an operand in both and there are no intervening Reads, Allocates, Writes, or Frees of v. A sequential execution is considered to be *well formed* if and only if

- operands to Read (resp., Write) statements are uncached (resp., cached) slow memory variables,
- operands to Allocate statements are uncached slow memory variables or fast memory variables[4],
- operands to Free statements are cached slow memory variables or fast memory variables,
- every Read, Allocate, Write, and Free statement is paired, and
- every Compute statement involving variable v interposes between paired Read/Allocate and Write/Free statements of v, i.e., each operand v in a Compute statements *resides in fast memory* before and after the statement.

Essentially, fast memory variables must be allocated and deallocated, either implicitly (Read/Write) or explicitly (Allocate/Free), while slow memory variables cannot be allocated/deallocated. Given the finite capacity of fast

[3] We will use the word 'copied' but our analysis does not require that a copy remains, e.g., exclusive caches.

[4] If a fast memory variable v in an *allocate*(v) statement already stores a cached slow memory variable, then we assume the system will first relocate the cached variable to an available location within fast memory)

memory, we need an additional assumption to ensure the memory operations are well-defined. Given a well-formed sequential execution $E = (e_1, \ldots, e_n)$, we define $footprint_i(E)$ to be the fast memory usage after executing statement e_i in the program, i.e.,

$$footprint_i(E) = \begin{cases} 0 & i = 0 \text{ or } i = n \\ footprint_{i-1}(E) + 1 & \text{if } e_i \text{ is a Read or Allocate statement,} \\ footprint_{i-1}(E) & \text{if } e_i \text{ is a Compute statement, and} \\ footprint_{i-1}(E) - 1 & \text{if } e_i \text{ is a Write or Free statement.} \end{cases}$$

Then E is said to be *M–fit* (for fast memory size M) if $\max_i footprint_i(E) \leq M$.

It is of practical interest to permit variables to reside in fast memory before and after the execution, e.g., to handle parallel code as described in the next paragraph; however, this violates our notion of well-formedness. Rather than redefine well-formedness to account for this possibility, we take a simpler approach: given an execution that is well formed except for the I ('input') and O ('output') variables which reside in fast memory before and after the execution, we insert up to M Reads and M Writes at the beginning and end of the execution so that all memory statements are paired, and then later reduce the lower bound (on Reads/Writes) by $I + O \leq 2M$.

Although we will establish our bounds first for a sequential execution, they also apply to parallel executions as explained in Section 4.6. We define a *parallel execution* as a set of P sequential executions, $\{E_1, E_2, \ldots, E_P\}$. In our parallel model, the global ('slow') memory for each processor is a subset of the union of the local ('fast') memories of the other processors. That is, for a given processor, each of its slow memory variables is really a fast memory variable for some other processor, and each of its fast memory variables is a slow memory variable for every other processor, unless it corresponds to a cached slow memory variable, in which case it is invisible to the other processors. (We could remove this last assumption by extending our model to distinguish between copies of a slow memory variable.) A parallel execution is well formed or (additionally) $\{M_1, \ldots, M_P\}$–fit if each of its serial executions E_i is well formed or (additionally) M_i–fit. Since well-formedness assumes no variables begin and end execution in fast/local memory, it seems impossible for there to be any nontrivial well-formed parallel execution. As mentioned above, we can always allow for a sequential execution with this property by inserting up to $I + O \leq 2M$ Reads/Writes, and later reducing the lower bound by this amount; we insert Reads/Writes in this manner to each sequential execution E_i in a parallel execution.

4.2 Relation to the geometric model

Recall from Section 2 our geometric model (2.2):

$$\begin{aligned} &\text{for all } \mathcal{I} \in \mathcal{Z} \subseteq \mathbb{Z}^d, \text{ in some order,} \\ &\quad \text{inner\_loop}(\mathcal{I}, (A_1, \ldots, A_m), (\phi_1, \ldots, \phi_m)) \end{aligned}$$

The subroutine inner_loop represents a given 'computation' involving arrays $A_1, \ldots, A_m$ referenced by corresponding subscripts $\phi_1(\mathcal{I}), \ldots, \phi_m(\mathcal{I})$; each $A_j \colon \phi_j(\mathbb{Z}^d) \to \{\text{variables}\}$ is an injection and each $\phi_j \colon \mathbb{Z}^d \to \mathbb{Z}^{d_j}$ is a $\mathbb{Z}$–affine map. Each *array variable* $A_1(\phi_1(\mathcal{I})), \ldots, A_m(\phi_m(\mathcal{I}))$ is accessed in each inner_loop($\mathcal{I}$), as an input, output, or both, perhaps depending on the iteration $\mathcal{I}$.

Given an execution, we assume we can discern the expression $A_j(\phi_j(\mathcal{I}))$ from any variable which represents an array variable in the geometric model. The execution may contain additional variables that act as *surrogates* (or copies) of the variables specified in the program text. As an extreme example, an execution could use an array A_j' as a surrogate for the array A_j in the computations, and then later set A_j to A_j'. In such examples, one can always associate each surrogate variable with the 'master' copy, and there is no loss of generality in our analysis to assume all variables are in fact the master copies.

We say a *legal sequential execution* of an instance of the geometric model is a sequential execution whose subsequence of Compute statements can be partitioned into contiguous chunks in one-to-one correspondence with $\mathcal{Z}$, and furthermore all m array variables $A_j(\phi_j(\mathcal{I}))$ appear as operands in the chunk corresponding to $\mathcal{I}$. Given a possibly overlapping partition $\mathcal{Z} = \bigcup_{i=1}^P \mathcal{Z}_i$, a *legal parallel execution* is a parallel execution $\{E_1, \ldots, E_p\}$ where each sequential execution E_i is legal with respect to loop iterations $\mathcal{I} \in \mathcal{Z}_i$. Legality restricts the set of possible concrete executions we consider (for a given instance of the geometric model), and in general is a necessary requirement for the lower bound to hold for all concrete executions. For example, transforming the classical algorithm for matrix multiplication into Strassen's algorithm (which can move asymptotically less data) is illegal, since it exploits the distributive property to interleave computations, and any resulting execution cannot be partitioned contiguously according to the original iteration space $\mathcal{Z}$. As another example, legality prevents *loop splitting*, an optimization which can invalidate the lower bound as discussed in Section 4.5.

We note that there are no assumptions about preserving dependencies in the original program or necessarily computing the correct answer. Restricting the set of executions further to ones that are correct may make the lower bound unattainable, but does not invalidate the bound.

4.3 Derivation of Communication Lower Bound

Now we present the more formal derivation of the communication lower bound, which was sketched in Section 2. The approach used here was introduced in [13] and generalized in [4]. Here we generalize it again to deal with the more complicated algorithms considered in this paper.

In this work, we are interested in the *asymptotic* communication complexity, in terms of parameters like the fast memory size M and the number of inner loop body iterations $|\mathcal{Z}|$. We treat the algorithm's other parameters, like the number of array references m, the dimension of the iteration space d, the array dimensions d_j, and the coefficients in subscripts ϕ_j, as constants. When discussing an upper bound F on the number of loop iterations doable with operands in a fast memory of M words, we assume $F = \Omega(M)$, since m is a constant. So, our asymptotic lower bound may hide a multiplicative factor $c(m, d, \{\phi_1, \ldots, \phi_m\})$, and this constant will also depend on whether any of the arrays $A_1, \ldots, A_m$ overlap. (Constants are important in practice, and will be addressed in Part 2 of this work.) Lastly, we assume $d > 0$, otherwise $|\mathcal{Z}| \leq 1$ and the given algorithm is already trivially communication optimal, and we assume $m > 0$, otherwise there are no arrays and thus no data movement.

Given an M–fit legal sequential execution $E = (e_1, \ldots, e_n)$, we proceed as follows:

1. Break E into *M–Read/Write segments* of consecutive statements, where each segment (except possibly the last one) contains exactly M Reads and/or Writes. Each segment (except the last) ends with the M^{th} Read/Write and the next segment starts with whatever statement follows. The last segment may have statements other than Reads/Writes at the end to complete the execution. (We will simply refer to these as Read/Write segments when M is clear from context.)

2. Independently, break E into *Compute segments* of consecutive statements so that the Compute statements within a segment correspond to the same iteration $\mathcal{I}$. (It will not matter that this does not uniquely define the first and last statements of a Compute segment.) Our assumption of a legal execution guarantees that there is one Compute segment per iteration $\mathcal{I}$. We associate each Compute segment with the (unique) Read/Write segment that contains the Compute segment's first Compute statement.

3. Using the limited availability of data in any one Read/Write segment, we will use Theorem 3.2 to establish an upper bound F on the number of complete Compute segments that can be executed during one Read/Write segment (see below). This is an upper bound on the number of complete loop iterations that can be executed.

4. Now, we can bound below the number of complete Read/Write segments by $\lfloor |\mathcal{Z}|/(F+1) \rfloor$. We add 1 to F to account for Compute segments that overlap two (or more) Read/Write segments. We need the floor function because the last Read/Write segment may not contain M Reads/Writes. (Since we are doing asymptotic analysis, $|\mathcal{Z}|$ can often be replaced by the total number of Compute statements.)

5. Finally we bound below the total number of Reads/Writes by the lower bound on the number of complete Read/Write segments times the number of Reads/Writes per such segment, minus the number of Reads/Writes we inserted to account for variables residing in fast memory before/after the execution:

$$M \left\lfloor \frac{|\mathcal{Z}|}{F+1} \right\rfloor - (I + O) = \Omega\left(\frac{M \cdot |\mathcal{Z}|}{F}\right) - O(M), \tag{4.1}$$

   where we have applied our asymptotic assumption $F = \Omega(M) = \omega(1)$.

To determine an upper bound F, we will use Theorem 3.2 to bound the amount of (useful) computation that can be done given only $O(M)$ array variables. First, we discuss how to ensure that only a fixed number of array variables is available during a single Read/Write segment.

Given an M–fit legal sequential execution, consider any M–Read/Write segment. There are at most M array variables in fast memory when the segment starts, at most M array variables are read/written during the segment, and at most M array variables remain in fast memory when the segment ends. If there are no Allocates of array variables, or if there are no Frees of array variables, then at most $2M$ distinct array variables appear in the segment (at most M may already reside in fast memory, and at most M more can be read or allocated). More generally, if there are no paired Allocate/Free statements of array variables, then at most $3M$ array variables appear in the segment (at most M already reside in fast memory, at most M can be read, and at most M can be allocated). However, if we allow array variables to be allocated and subsequently freed, then it is possible to have an unbounded number of array variables contributing to computation in the same segment; this can occur in practice and we give concrete examples in the following section. Thus, we need an additional assumption in order to obtain a lower bound that is valid for all executions.

We will assume that the execution contains no paired Allocate/Free statements of array variables. However, we note that of these paired statements, we only need to avoid the ones where both statements occur within a given Read/Write segment; e.g., one could remove the preceding assumption by proving that at least M Read/Write statements (of variables besides v) interpose every paired Allocate/Free of an array variable v. (This weaker assumption is equivalent to an assumption in [4, Section 2] that there are no 'R2/D2 operands.')

The communication lower bound is now a straightforward application of Theorem 3.2.

Theorem 4.1. *Consider an algorithm in the geometric model of* (2.2) *in Section 2, and consider any M–fit, legal sequential execution which contains no paired Allocate/Free statements of array variables. If the linear constraints* (3.1) *of Theorem 3.2 are feasible, then for sufficiently large $|\mathcal{Z}|$, the number of Reads/Writes in the execution is $\Omega(|\mathcal{Z}|/M^{s_{\text{HBL}}-1})$, where s_{HBL} is the minimum value of $\sum_{j=1}^{m} s_j$ subject to* (3.1).

Proof. The bound F may be derived from Theorem 3.2 as follows. If $E \subseteq \mathcal{Z}$ is the (finite) set of complete inner loop body iterations executed during a Read/Write segment, then we may bound $|E| \le \prod_{j=1}^{m} |\phi_j(E)|^{s_j}$ for any $s = (s_1, \ldots, s_m)$ satisfying (3.1). By our assumptions, each M–Read/Write segment has at most $3M$ distinct array variables whose values reside in fast memory. This implies that $\max_j |\phi_j(E)| \le 3M$, since we allow arrays to alias[5]. So, $|E| \le \prod_{j=1}^{m} (3M)^{s_j} = (3M)^{\sum_{j=1}^{m} s_j}$. Since this bound applies for any s satisfying (3.1), we choose an s minimizing $\sum_{j=1}^{m} s_j$, obtaining the tightest bound $|E| \le (3M)^{s_{\text{HBL}}} =: F$. The communication lower bound is $\Omega(M \cdot |\mathcal{Z}|/F) - O(M)$; if we assume $|\mathcal{Z}| = \omega(F)$, i.e., the iteration space is 'sufficiently large,' then we obtain $\Omega(|\mathcal{Z}|/M^{s_{\text{HBL}}-1})$ as desired. □

When $|\mathcal{Z}| = \Theta(F)$, i.e., the problem (iteration space) is not sufficiently large, the lower bound becomes $\Omega(M) - O(M)$, so the subtractive $O(M)$ term may dominate and lead to zero communication; this is increasingly likely as the ratio $|\mathcal{Z}|/F$ goes to zero. The parallel case also demonstrates this behavior in the 'strong scaling' limit, when the problem is decomposed to the point that each processor's working set fits in their local memory (see Section 4.6). In the regime $|\mathcal{Z}| = O(F)$, a *memory-independent* lower bound [3] provides more insight than the bound above. Let W be the (unknown) number of Reads/Writes performed, and let I and O be the numbers of input/output variables residing in fast memory before/after execution. Then

$$|\mathcal{Z}| \le \prod_j |\phi_j(\mathcal{Z})|^{s_j} \le (\max_j |\phi_j(\mathcal{Z})|)^{s_{\text{HBL}}} \le (W + I + O)^{s_{\text{HBL}}},$$

and we have $W \ge |\mathcal{Z}|^{1/s_{\text{HBL}}} - (I + O)$; see Section 7.3 for further discussion. While this bound applies for any $|\mathcal{Z}| \ge 1$, the memory-*dependent* bound in Theorem 4.1 is tighter when $|\mathcal{Z}|$ is sufficiently large; when $|\mathcal{Z}|$ is not so large, the memory-independent bound may be tighter. In Part 1 of this work, we are interested in lower bounds for large problems, and so only consider the memory-dependent bound. In Part 2, we will revisit the case of smaller problems when we discuss the constants hidden in our asymptotic bounds.

4.4 Examples

We give four examples to demonstrate why our assumptions in Theorem 4.1 are necessary. Then, we discuss how one can sometimes deal with the presence of imperfectly nested loops, paired Allocate/Free statements, and an infeasible linear program to compute a useful lower bound; we give an example of this approach.

Example: Modified Matrix Multiplication with paired Allocates/Frees (I). The following simple modification of matrix multiplication demonstrates how paired Allocate/Frees can invalidate our lower bound.

```
**for** $i_1 = 1 : N$,    **for** $i_2 = 1 : N$,    **for** $i_3 = 1 : N$,
    $C(i_1, i_2) = A(i_1, i_3) \cdot B(i_3, i_2)$
    $t = t + C(i_1, i_2)^7$
```

[5] If the arrays do not alias, then the tighter constraint $\sum_{j=1}^{m} |\phi_j(E)| \le 3M$ holds instead, although our bound here is still valid. We will discuss tightening our bounds for nonaliasing arrays in Part 2 of this work.

Suppose we know the arrays do not alias each other. Clearly C only depends on the data when $i_3 = N$, but we need to do all N^3 multiplications to compute t correctly. But the same analysis from Section 3 applies to these loops as to matrix multiplication, suggesting a sequential communication lower bound of $\Omega(N^3/M^{1/2})$. However, it is clearly possible to execute all N^3 iterations moving only $O(N^3/M + N^2)$ words, by hoisting the (unblocked) i_3 loop outside and blocking the i_1 and i_2 loops by $M/2$, doing $(M/2)^2$ multiplications in a Read/Write segment using $M/2$ entries each of $A(\cdot, i_3)$ and $B(i_3, \cdot)$, and (over)writing the values $C(i_1, i_2)$ to a single location in fast memory, which is repeatedly Allocated and Freed. Only when $i_3 = N$ would $C(i_1, i_2)$ actually be written to slow memory. So in this case there are a total of $N^3 - N^2$ paired Allocates/Frees, corresponding to the overwritten $C(i_1, i_2)$ operands. △

Example: Modified Matrix Multiplication with Paired Allocate/Frees (II). This example also demonstrates how paired Allocate/Frees can invalidate our lower bound. Consider the following code:

$$\begin{aligned}&\textbf{for } i_1 = 1 : N,\ \textbf{for } i_2 = 1 : N, \quad A(i_1, i_2) = e^{2\pi i (i_1-1)(i_2-1)/N}\\ &\textbf{for } i_1 = 1 : N,\ \textbf{for } i_2 = 1 : N,\ \textbf{for } i_3 = 1 : N, \quad C(i_1, i_2) \mathrel{+}= A(i_1, i_3) \cdot B(i_3, i_2)\end{aligned}$$

Again, suppose we know that the arrays do not alias each other. Toward a lower bound, we ignore the initialization of A (first loop nest) and only look at the second loop nest, a matrix multiplication. However, by computing entries of A on-the-fly from $\mathcal{I}$ and discarding them (i.e., Allocating/Freeing them), one can beat the lower bound of $\Omega(N^3/M^{1/2})$ words for matrix multiplication. That is, by hoisting the (unblocked) i_2 loop outside and blocking the i_1 and i_3 loops by $M/2$, and finally writing each $A(i_1, i_3)$ to slow memory when $i_2 = N$, we can instead move $O(N^3/M + N^2)$ words. So, there are $N^3 - N^2$ possible paired Allocate/Frees. △

Example: Infeasibility. This example demonstrates how infeasibility of the linear constraints (3.1) of Theorem 3.2 can invalidate our lower bound. Consider the following code:

$$\begin{gathered}\textbf{for } i_1 = 1 : N_1, \quad \textbf{for } i_2 = 1 : N_2,\\ A(i_1) = \text{func}(A(i_1))\end{gathered}$$

It turns out the linear constraints (3.1) are infeasible, so we cannot apply Theorem 3.2 to find a bound on data reuse of the form $M^{s_{\text{HBL}}-1}$. It is easy to see that only $2N_1$ Reads and Writes of A are needed to execute the inner loop $N_1 \cdot N_2$ times, i.e., unbounded (N_2–fold) data reuse is possible. In general, infeasibility suggests that the number of available array variables in fast memory during a Read/Write segment is constrained only by the iteration space $\mathcal{Z}$. (We will explore infeasibility further in Sections 6.2.1 and 7.1.1.)

While infeasibility may be sufficient for there to be an unbounded number of array variables in a Read/Write segment, the previous two examples show that it is not necessary, since their linear programs are feasible. We will be more concrete about this relationship between infeasibility and unbounded data reuse in Part 2. △

Example: Loop interleaving. This example demonstrates how an execution which interleaves the inner loop bodies (an *illegal* execution) can invalidate our lower bound; see also Section 4.5. We will see in Section 4.5 that the lower bound for each split loop is no larger than the lower bound for the original loop. Consider splitting the two lines of the inner loop body in the Complicated Code example (see Section 1) into two disjoint loop nests (each over $\mathcal{I} \in \{1, \ldots, N\}^6$). We assume func_1 and func_2 do not modify their arguments, and that the arrays do not alias — the two lines share only read accesses to one array, A_3, so correctness is preserved. As will be seen later by using Theorem 7.1, the resulting two loop nests have lower bounds $\Omega(N^6/M^{3/2})$ and $\Omega(N^6/M^2)$, resp., both better than the $\Omega(N^6/M^{8/7})$ of the original, and both these lower bounds are attainable. △

Theorem 4.1 is enough for many direct linear algebra computations such as (dense or sparse) LU decomposition, which do not have paired Allocate/Frees, but not all algorithms for the QR decomposition or eigenvalue problems, which can potentially have large numbers of paired Allocates/Frees (see [4]). We can often deal with interleaving iterations, paired Allocates/Frees, and infeasibility of (3.1) by *imposing Reads and Writes* [4, Section 3.4]: we modify the algorithm to add ("impose") Reads and Writes of array variables which are allocated/freed or repeatedly overwritten, apply the lower bound from Theorem 4.1, and then subtract the number of imposed Reads and Writes to get the final lower bound. (Note that we have already used a similar technique, above, to allow an execution to begin/end with a nonzero fast memory footprint.) We give an example of this approach (see also [4, Corollary 5.1]).

Example: Matrix Powering (Part 1/4). Consider computing $B = A^k$ using the following code, shown (for simplicity) for odd k and initially $B = A$:

```
// Original Code
for $i_1 = 1 : \lfloor k/2 \rfloor$
    $C = A \cdot B$
    $B = A \cdot C$
```

In order to get the correct answer, we assume the arrays do not alias each other. Under our asymptotic assumption that the number m of arrays accessed in the inner loop is constant with respect to $|\mathcal{Z}|$, one cannot simply model the code as a 1–deep loop nest (over i_1), since each entry of A, B, C would be considered an array. Considering instead the scalar multiply/adds as the 'innermost loop,' we violate an assumption of the geometric model (2.2): since $C = A \cdot B$ needs to be (partially) completed before $B = A \cdot C$ can be computed, the innermost loops necessarily interleave. Toward obtaining a lower bound, we could try to apply our theory to a perfectly nested subset of the code, omitting $B = A \cdot C$; as will be shown in part 2/4 of this example (see Section 6), the corresponding linear constraints in (3.1) are then infeasible, violating another assumption of Theorem 4.1. The same issue arises if we omit $C = A \cdot B$; furthermore, neither modification prevents the possibility of paired Allocates/Frees of array variables of B, C. We deal with all three violations by rewriting the algorithm in the model (2.2) and imposing Reads and Writes of all intermediate powers A^i with $1 < i_1 < k$, so at most $2(k-2)N^2$ Reads and Writes altogether. This can be expressed by using one array $\hat{A}$ with three subscripts, where $\hat{A}(1, i_2, i_3) \coloneqq A(i_2, i_3)$ and all other entries are zero-initialized.

```
// Modified Code (Imposed Reads and Writes)
for $i_1 = 2 : k$,    for $i_2 = 1 : N$,    for $i_3 = 1 : N$,    for $i_4 = 1 : N$,
    $\hat{A}(i_1, i_2, i_3) \mathrel{+}= \hat{A}(1, i_2, i_4) \cdot \hat{A}(i_1 - 1, i_4, i_3)$
```

This code clearly matches the geometric model (2.2), and admits legal executions (which would be interleaving executions of the original code). As will be shown in part 3/4 of this example (see Section 6), the linear constraints (3.1) are now feasible, and the resulting exponent from Theorem 4.1 will be $s_{\mathrm{HBL}} = 3/2$, so if the matrices are all N–by–N, the lower bound of the original program will be $\Omega(kN^3/M^{s_{\mathrm{HBL}}-1}) - 2(k-2)N^2$. For simplicity, suppose that this can be rewritten as $\Omega(kN^3/M^{1/2} - kN^2)$. So we see that when the matrices are small enough to fit in fast memory M, i.e., $N \leq M^{1/2}$, the lower bound degenerates to 0, which is the best possible lower bound which is also proportional to the total number of loop iterations $|\mathcal{Z}| = (k-1)N^3$. But for larger N, the lower bound simplifies to $\Omega(kN^3/M^{1/2})$ which is in fact attained by the natural algorithm that does $k-1$ consecutive (optimal) N–by–N matrix multiplications.

This example also illustrates that our results will only be of interest for sufficiently large problems, certainly where the floor function in the lower bound (4.1) is at least 1. △

The above approach covers many but not all algorithms of interest. We refer to reader to [4, Sections 3.4 and 5] for more examples of imposing Reads and Writes, and [4, Section 4] on orthogonal matrix factorizations for an important class of algorithms where a subtler analysis is required to deal with paired Allocates/Frees.

Imposing Reads and Writes may fundamentally alter the program, so the lower bound obtained for the modified code need not apply to the original code. In the example above, one could reorder[6] the original code to

```
for $i_2 = 1 : N$,    for $i_3 = 1 : N$,    for $i_4 = 1 : N$,    for $i_1 = 1 : \lfloor k/2 \rfloor$,
    $C(i_2, i_3) \mathrel{+}= A(i_2, i_4) \cdot B(i_4, i_3)$
    $B(i_2, i_3) \mathrel{+}= A(i_2, i_4) \cdot C(i_4, i_3)$.
```

Recall that our lower bounds are valid for any reordering of the iteration space, correct or otherwise. Since i_1 does not appear in the inner loop body, data movement is independent of k. In fact, this code moves $O(N^3/M^{1/2})$ words, asymptotically beating the lower bound for the modified code (with imposed Reads/Writes). For another example, see [4, Section 5.1.3]. In Part 2, we will present an alternative to imposing Reads/Writes which can yield a (nontrivial) lower bound on the original code: roughly speaking, one ignores the loops whose indices do not appear in any subscripts (e.g., i_1, above). While this approach will eliminate infeasibility, paired Allocates/Frees may still arise and imposing Reads/Writes may still be necessary.

[6] For simplicity, and without reducing data movement, this code performs $(k-1)N^2$ additional + operations.

4.5 Loop Splitting Can Only Help

Here we show that loop splitting can only reduce (improve) the communication lower bound expressed in Theorem 4.1. More formally, we state this as the following.

Theorem 4.2. *Suppose we have an algorithm satisfying the hypotheses of Theorem 4.1, with lower bound determined by the value s_{HBL}. Also suppose that this algorithm can be rewritten as $t > 1$ consecutive (disjoint) loop nests, where each has the same iteration space as the original algorithm but accesses a subset of the original array entries, and each satisfies the hypotheses of Theorem 4.1, leading to exponents $s_{\text{HBL},i}$ for $i \in \{1, \ldots, t\}$. Then each $s_{\text{HBL},i} \geq s_{\text{HBL}}$, i.e., the communication lower bound for each new loop nest is asymptotically at least as small as the communication lower bound for the original algorithm.*

Proof. By our assumptions, s_{HBL} is the minimum value of $\sum_{j=1}^m s_j$ where $s \in [0,1]^m$ satisfies the constraints (3.1). We need to prove that if we replace these constraints by

$$\operatorname{rank}(H) \leq \sum_{j \in L_i} s_j \operatorname{rank}(\phi_j(H)) \qquad \text{for all subgroups } H \leq \mathbb{Z}^d, \tag{4.2}$$

where L_i is any given nonempty subset of $\{1, \ldots, m\}$ corresponding to the i^{th} (of t) new loop nest, then the minimum value of $s_{\text{HBL},i} = \sum_{j \in L_i} s_j$ for any s satisfying (4.2) must satisfy $s_{\text{HBL},i} \geq s_{\text{HBL}}$. We proceed by replacing both sets of constraints by a common finite set, yielding linear programs, and then use duality.

As mentioned immediately after the statement of Theorem 3.2, even though there are infinitely many subgroups $H \leq \mathbb{Z}^d$, there are only finitely many possible values of each rank, so we may choose a finite set $\mathcal{S}$ of subgroups $H \leq \mathbb{Z}^d$ that yield all possible constraints (3.1). Similarly, there is a finite set of subgroups $\mathcal{S}_i$ that yields all possible constraints (4.2). Now let $\mathcal{S}^* = \mathcal{S} \cup \mathcal{S}_i$; we may therefore replace both (3.1) and (4.2) by a finite set of constraints, for the subgroups $H \in \mathcal{S}^* = \{H_1, \ldots, H_g\}$.

This lets us rewrite (3.1) as the linear program of minimizing $s_{\text{HBL}} = \sum_{j=1}^m s_j = 1_m^T \cdot s$ subject to $s^T \cdot \Delta \geq r^T$, where $s = (s_1, \ldots, s_m)^T$, 1_m is a column vector of ones, $r = (\operatorname{rank}(H_1), \ldots, \operatorname{rank}(H_g))^T$, and Δ is m–by–g with $\Delta_{ij} = \operatorname{rank}(\phi_i(H_j))$. Assuming without loss of generality that the constraints L_i are the first constraints $L_i = \{1, 2, \ldots, |L_i|\}$, we may similarly rewrite (4.2) as the linear program of minimizing $s_{\text{HBL},i} = \sum_{j=1}^{|L_i|} s_j = 1_{|L_i|}^T \cdot s_{L_i}$ subject to $s_{L_i}^T \cdot \Delta_{L_i} \geq r^T$, where Δ_{L_i}, s_{L_i} and $1_{|L_i|}$ consist of the first $|L_i|$ rows of Δ, s and 1_m, resp. The duals of these two linear programs, which have the same optimal values as the original linear programs, are

$$\text{maximize } \; r^T \cdot x \; \text{ subject to } \; \Delta \cdot x \leq 1_m \tag{4.3}$$

and

$$\text{maximize } \; r^T \cdot x_{L_i} \; \text{ subject to } \; \Delta_{L_i} \cdot x_{L_i} \leq 1_{|L_i|} \tag{4.4}$$

resp., where both x and x_{L_i} have dimension g. It is easy to see that (4.4) is maximizing the same quantity as (4.3), but subject to a subset of the constraints of (4.3), so its optimum value, $s_{\text{HBL},i}$, must be at last as large as the other optimum, s_{HBL}. □

While loop splitting appears to always be worth attempting, in practice data dependencies limit our ability to perform this optimization; we will discuss the practical aspects of loop splitting further in Part 2 of this work.

4.6 Generalizing the machine model

Earlier we said the reader could think of a sequential algorithm where the fast memory consists of a cache of M words, and slow memory is the main memory. In fact, the result can be extended to the following situations:

Multiple levels of memory. If a sequential machine has a memory hierarchy, i.e., multiple levels of cache (most do), where data may only move between adjacent levels, and arithmetic done only on the "top" level, then it is of interest to bound the data transferred between every pair of adjacent levels, say i and $i+1$, where i is higher (faster and closer to the arithmetic unit) than $i+1$. In this case we apply our model with M representing the total memory available in levels 1 through i, typically an increasing function of i.

Homogeneous parallel processors. We call a parallel machine *homogeneous* if it consists of P identical processors, each with its own memory, connected over some kind of network. For any processor, fast memory is the memory it owns, and Reads and Writes refer to moving data over the network from or to the other processors' memories. Recalling notation from above, consider an $\{M, \ldots, M\}$–fit, legal parallel execution $(E_1, \ldots, E_P)$, i.e., a set of M–fit, legal sequential executions E_i (with corresponding $\mathcal{Z}_i \subseteq \mathcal{Z}$); assuming there are no paired Allocate/Frees, we can apply Theorem 4.1 to each and obtain the lower bound of $\Omega(|\mathcal{Z}_i|/M^{s_{\mathrm{HBL}}-1})$ words moved. As argued in [3], to minimize computational costs, necessarily at least one of the processors performs $|\mathcal{Z}|/P$ (distinct) iterations; since a lower bound for this processor is a lower bound for the critical path, we take $|\mathcal{Z}_i| = |\mathcal{Z}|/P$, and obtain the lower bound of $\Omega(|\mathcal{Z}|/(PM^{s_{\mathrm{HBL}}-1}))$ words moved (along the critical path).

We recall that Theorem 4.1 makes an asymptotic assumption on $|\mathcal{Z}|$; having replaced $|\mathcal{Z}|$ by $|\mathcal{Z}|/P$ in the parallel case, this assumption becomes $|\mathcal{Z}|/P = \omega(M^\sigma)$. When this assumption fails (e.g., $|\mathcal{Z}|$ is small or P is large), it may be possible for each processor to store its entire working set locally, with no communication needed. As in the sequential case (see above), one may obtain a memory-independent lower bound $W = (|\mathcal{Z}|/P)^{1/s_{\mathrm{HBL}}} - (I + O)$, which may be tighter in this regime; this result generalizes [3, Lemma 2.3]. Interestingly, one can show that under certain assumptions on I and O (e.g., only one copy of the inputs/outputs is permitted at the beginning/end), while the communication cost continues to decrease with P (up to the natural limit $P = |\mathcal{Z}|$), it fails to strong-scale perfectly when the assumption on $|\mathcal{Z}|/P$ breaks; e.g., see [3, Corollary 3.2]. We will discuss strong scaling limits further in Part 2 of this work.

One may also ask what value of M to use for each processor. Suppose that each processor has $M_{\max}$ words of fast memory, and that the total problem size of all the array entries accessed is M_{array}. So if each processor gets an equal share of the data we use $M = M_{\mathrm{array}}/P \le M_{\max}$. But the lower bound may still apply, and be smaller, if M is larger than M_{array}/P (but at most $M_{\max}$). In some cases algorithms are known that attain these smaller lower bounds (e.g., matrix multiplication in [20]), i.e., replicating data can reduce communication.

In Section 7.3, we discuss attainability of these parallel lower bounds, and reducing communication by replicating data.

Hierarchical parallel processors. The simplest possible hierarchical machine is the sequential one with multiple levels of memory discussed above. But real parallel machines are similar: each processor has its own memory organized in a hierarchy. So just as we applied our lower bound to measure memory traffic between levels i and $i+1$ of cache on a sequential machine, we can similarly analyze the memory hierarchy on each processor in a parallel machine.

Heterogeneous machines. Finally, people are building heterogeneous parallel machines, where the various processors, memories, and interconnects can have different speeds or sizes. Since minimizing the total running time means minimizing the time when the last processor finishes, it may no longer make sense to assign an equal fraction $|\mathcal{Z}|/P$ of the work and equal subset of memory M to each processor. Since our lower bounds apply to each processor independently, they can be used to formulate an optimization problem that will give the optimal amount of work to assign to each processor [2].

5 (Un)decidability of the communication lower bound

In Section 3, we proved Theorem 3.2, which tells us that the exponents $s \in [0,1]^m$ satisfy the inequalities (3.1), i.e.,

$$\operatorname{rank}(H) \le \sum_{j=1}^{m} s_j \operatorname{rank}(\phi_j(H)) \qquad \text{for all subgroups } H \le \mathbb{Z}^d,$$

precisely when the desired bound (3.2) holds. If so, then following Theorem 4.1, the sum of these exponents $s_{\mathrm{HBL}} \coloneqq \sum_{j=1}^m s_j$ leads to a communication lower bound of $\Omega(|\mathcal{Z}|/M^{s_{\mathrm{HBL}}-1})$. Since our goal is to get the tightest bound, we want to minimize $\sum_{j=1}^m s_j$ subject to (3.1) and $s \in [0,1]^m$. In this section, we will discuss computing the set of inequalities (3.1), in order to write down and solve this minimization problem.

We recall from Section 3.3 that the feasible region for s (defined by these inequalities) is a convex polytope $\mathcal{P} \subseteq [0,1]^m$ with finitely many extreme points. While $\mathcal{P}$ is uniquely determined by its extreme points, there may be many sets of inequalities which specify $\mathcal{P}$; thus, it suffices to compute any such set of inequalities, rather than the specific set (3.1). This distinction is important in the following discussion. In Section 5.1, we show that there is an effective algorithm to determine $\mathcal{P}$. However, it is not known whether it is decidable to compute the set of inequalities (3.1) which define $\mathcal{P}$ (see Section 5.2). In Section 5.3, we discuss two approaches for approximating $\mathcal{P}$, providing upper and lower bounds on the desired s_{HBL}.

5.1 An Algorithm Which Computes $\mathcal{P}$

We have already shown in Lemma 3.14 that the polytope $\mathcal{P}$ is unchanged when we embed the groups $\mathbb{Z}^d$ and $\mathbb{Z}^{d_j}$ into the vector spaces $\mathbb{Q}^d$ and $\mathbb{Q}^{d_j}$ and consider the homomorphisms ϕ_j as $\mathbb{Q}$–linear maps. Thus it suffices to compute the polytope $\mathcal{P}$ corresponding to the inequalities

$$\dim(V) \le \sum_{j=1}^{m} s_j \dim(\phi_j(V)) \qquad \text{for all subgroups } V \le \mathbb{Q}^d.$$

Indeed, combined with the constraints $s \in [0,1]^m$, this is the hypothesis (3.8) of Theorem 3.10 in the case $\mathbb{F} = \mathbb{Q}$.

We will show how to compute $\mathcal{P}$ in the case $\mathbb{F} = \mathbb{Q}$; for the remainder of this section, V and V_j denote finite-dimensional vector spaces over $\mathbb{Q}$, and ϕ_j denotes a $\mathbb{Q}$–linear map. We note that the same reasoning applies to any countable field $\mathbb{F}$, provided that elements of $\mathbb{F}$ and the field operations are computable.

Theorem 5.1. *There exists an algorithm which takes as input any vector space HBL datum over the rationals, i.e., $\mathbb{F} = \mathbb{Q}$, and returns as output both a list of finitely many linear inequalities over $\mathbb{Z}$ which jointly specify the associated polytope $\mathcal{P}(V,(V_j),(\phi_j))$, and a list of all extreme points of $\mathcal{P}(V,(V_j),(\phi_j))$.*

Remark 5.2. *The algorithm we describe below to prove Theorem 5.1 is highly inefficient, because of its reliance on a search of arbitrary subspaces of V. A similar algorithm was sketched in [25] for computing the polytope in [6, Theorem 2.1], a result related to Theorem 3.10 in the case $\mathbb{F} = \mathbb{R}$ but where the vector spaces V, V_j are endowed with additional inner-product structure. That algorithm searches a smaller collection of subspaces, namely the lattice generated by the nullspaces of $\phi_1, \ldots, \phi_m$. In Part 2 of this work (see Section 8) we will show that it suffices to search this same lattice in our case; this may significantly improve the efficiency of our approach. Moreover, this modification will also allow us to relax the requirement that $\mathbb{F}$ is countable, although this is not necessary for our application.*

The proof of Theorem 5.1 is built upon several smaller results.

Lemma 5.3. *There exists an algorithm which takes as input a finite-dimensional vector space V over $\mathbb{Q}$, and returns a list of its subspaces. More precisely, this algorithm takes as input a finite-dimensional vector space V and a positive integer N, and returns as output the first N elements W_i of a list $(W_1, W_2, \ldots)$ of all subspaces of V. This list is independent of N. Each subspace W is expressed as a finite sequence $(d; w_1, \ldots, w_d)$ where $d = \dim(W)$ and $\{w_i\}$ is a basis for W.*

Proof. Generate a list of all nonempty subsets of V having at most $\dim(V)$ elements. Test each subset for linear independence, and discard all which fail to be independent. Output a list of those which remain. □

We do not require this list to be free of redundancies.

Lemma 5.4. *For any positive integer m, there exists an algorithm which takes as input a finite set of linear inequalities over $\mathbb{Z}$ for $s \in [0,1]^m$, and returns as output a list of all the extreme points of the convex subset $\mathcal{P} \subseteq [0,1]^m$ specified by these inequalities.*

Proof. To the given family of inequalities, adjoin the $2m$ inequalities $s_j \ge 0$ and $-s_j \ge -1$. $\mathcal{P}$ is the convex polytope defined by all inequalities in the resulting enlarged family. Express these inequalities as $\langle s, v_\alpha \rangle \ge c_\alpha$ for all $\alpha \in A$, where A is a finite nonempty index set.

An arbitrary point $\tau \in \mathbb{R}^m$ is an extreme point of $\mathcal{P}$ if and only if (i) there exists a set B of indices α having cardinality m, such that $\{v_\beta : \beta \in B\}$ is linearly independent and $\langle \tau, v_\beta \rangle = c_\beta$ for all $\beta \in B$, and (ii) τ satisfies $\langle \tau, v_\alpha \rangle \ge c_\alpha$ for all $\alpha \in A$.

Create a list of all subsets $B \subset A$ with cardinality equal to m. There are finitely many such sets, since A itself is finite. Delete each one for which $\{v_\beta : \beta \in B\}$ is not linearly independent. For each subset B not deleted, compute the unique solution τ of the system of equations $\langle \tau, v_\beta \rangle = c_\beta$ for all $\beta \in B$. Include τ in the list of all extreme points, if and only if τ satisfies $\langle \tau, v_\alpha \rangle \ge c_\alpha$ for all $\alpha \in A \setminus B$. □

Proposition 5.5. *There exists an algorithm which takes as input a vector space HBL datum $(V,(V_j),(\phi_j))$, an element $t \in [0,1]^m$, and a subspace $\{0\} < W < V$ which is critical with respect to t, and determines whether $t \in \mathcal{P}(V,(V_j),(\phi_j))$.*

Theorem 5.1 and Proposition 5.5 will be proved inductively in tandem, according to the following induction scheme. The proof of Theorem 5.1 for HBL data in which V has dimension n, will rely on Proposition 5.5 for HBL data in which V has dimension n. The proof of Proposition 5.5 for HBL data in which V has dimension n and there are m subspaces V_j, will rely on Proposition 5.5 for HBL data in which V has dimension strictly less than n, on Theorem 5.1 for HBL data in which V has dimension strictly less than n, and also on Theorem 5.1 for HBL data in which V has dimension n and the number of subspaces V_j is strictly less than m. Thus there is no circularity in the reasoning.

Proof of Proposition 5.5. Let $(V, (V_j), (\phi_j))$ and t, W be given. Following Notation 3.23, consider the two HBL data $(W, (\phi_j(W)), (\phi_j\big|_W))$ and $(V/W, (V_j/\phi_j(W)), ([\phi_j]))$, where $[\phi_j]\colon V/W \to V_j/\phi_j(W)$ are the quotient maps. From a basis for V, bases for V_j, a basis for W, and corresponding matrix representations of ϕ_j, it is possible to compute the dimensions of, and bases for, V/W and $V_j/\phi_j(W)$, via row operations on matrices. According to Lemma 3.25, $t \in \mathcal{P}(V, (V_j), (\phi_j))$ if and only if $t \in \mathcal{P}(W, (\phi_j(W)), (\phi_j\big|_W)) \cap \mathcal{P}(V/W, (V_j/\phi_j(W)), ([\phi_j]))$.

Because $0 < W < V$, both $W, V/W$ have dimensions strictly less than the dimension of V. Therefore by Theorem 5.1 and the induction scheme, there exists an algorithm which computes both a finite list of inequalities characterizing $\mathcal{P}(W, (\phi_j(W)), (\phi_j\big|_W))$, and a finite list of inequalities characterizing $\mathcal{P}(V/W, (V_j/\phi_j(W)), ([\phi_j]))$. Testing each of these inequalities on t determines whether t belongs to these two polytopes, hence whether t belongs to $\mathcal{P}(V, (V_j), (\phi_j))$. □

Lemma 5.6. *Let HBL datum $(V, (V_j), (\phi_j))$ be given. Let $i \in \{1, 2, \ldots, m\}$. Let $s \in [0,1]^m$ and suppose that $s_i = 1$. Let $V' = \{x \in V : \phi_i(x) = 0\}$ be the nullspace of ϕ_i. Define $\widehat{s} \in [0,1]^{m-1}$ to be $(s_1, \ldots, s_m)$ with the i^{th} coordinate deleted. Then $s \in \mathcal{P}(V, (V_j), (\phi_j))$ if and only if $\widehat{s} \in \mathcal{P}(V', (V_j)_{j\neq i}, (\phi_j\big|_{V'})_{j\neq i})$.*

Proof. For any subspace $W \le V'$, since $\dim(\phi_i(W)) = 0$,

$$\sum_j s_j \dim(\phi_j(W)) = \sum_{j\neq i} s_j \dim(\phi_j(W)).$$

So if $s \in \mathcal{P}(V, (V_j), (\phi_j))$ then $\widehat{s} \in \mathcal{P}(V', (V_j)_{j\neq i}, (\phi_j\big|_{V'})_{j\neq i})$.

Conversely, suppose that $\widehat{s} \in \mathcal{P}(V', (V_j)_{j\neq i}, (\phi_j\big|_{V'})_{j\neq i})$. Let W be any subspace of V. Write $W = W'' + (W \cap V')$ where the subspace $W'' \le V$ is a supplement to $W \cap V'$ in W, so that $\dim(W) = \dim(W'') + \dim(W \cap V')$. Then

$$\begin{aligned}\sum_j s_j \dim(\phi_j(W)) &= \dim(\phi_i(W)) + \sum_{j\neq i} s_j \dim(\phi_j(W)) \\ &\ge \dim(\phi_i(W'')) + \sum_{j\neq i} s_j \dim(\phi_j(W \cap V')) \\ &\ge \dim(W'') + \dim(W \cap V');\end{aligned}$$

$\dim(\phi_i(W'')) = \dim(W'')$ because ϕ_i is injective on W''. So $s \in \mathcal{P}(V, (V_j), (\phi_j))$. □

To prepare for the proof of Theorem 5.1, let $\mathcal{P}(V, (V_j), (\phi_j))$ be given. Let $(W_1, W_2, W_3, \ldots)$ be the list of subspaces of V produced by the algorithm of Lemma 5.3. Let $N \ge 1$. To each index $\alpha \in \{1, 2, \ldots, N\}$ is associated a linear inequality $\sum_{j=1}^m s_j \dim(\phi_j(W_\alpha)) \ge \dim(W_\alpha)$ for elements $s \in [0,1]^m$, which we encode by an $(m+1)$–tuple $(v(W_\alpha), c(W_\alpha))$; the inequality is $\langle s, v(W_\alpha)\rangle \ge c(W_\alpha)$. Define $\mathcal{P}_N \subseteq [0,1]^m$ to be the polytope defined by this set of inequalities.

Lemma 5.7.

$$\mathcal{P}_N \supseteq \mathcal{P}(V, (V_j), (\phi_j)) \qquad \textit{for all } N. \tag{5.1}$$

Moreover, there exists a positive integer N such that $\mathcal{P}_M = \mathcal{P}(V, (V_j), (\phi_j))$ for all $M \ge N$.

Proof. The inclusion holds for every N, because the set of inequalities defining $\mathcal{P}_N$ is a subset of the set defining $\mathcal{P}(V, (V_j), (\phi_j))$.

$\mathcal{P}(V, (V_j), (\phi_j))$ is specified by some finite set of inequalities, each specified by some subspace of V. Choose one such subspace for each of these inequalities. Since (W_α) is a list of all subspaces of V, there exists M such that each of these chosen subspaces belongs to $(W_\alpha : \alpha \le M)$. □

Lemma 5.8. *Let $m \ge 2$. If s is an extreme point of $\mathcal{P}_N$, then either $s_j \in \{0,1\}$ for some $j \in \{1, 2, \ldots, m\}$, or there exists $\alpha \in \{1, 2, \ldots, N\}$ for which W_α is critical with respect to s and $0 < \dim(W_\alpha) < \dim(V)$.*

In the following argument, we say that two inequalities $\langle s, v(W_\alpha)\rangle \geq c(W_\alpha)$, $\langle s, v(W_\beta)\rangle \geq c(W_\beta)$ are distinct if they specify different subsets of $\mathbb{R}^m$.

Proof. For any extreme point s, equality must hold in at least m genuinely distinct inequalities among those defining $\mathcal{P}_N$. These inequalities are of three kinds: $\langle s, v(W_\alpha)\rangle \geq c(W_\alpha)$ for $\alpha \in \{1, 2, \ldots, N\}$, $s_j \geq 0$, and $-s_j \geq -1$, with $j \in \{1, 2, \ldots, m\}$. If $W_\beta = \{0\}$ then W_β specifies the tautologous inequality $\sum_j s_j \cdot 0 = 0$, so that index β can be disregarded.

If none of the coordinates s_j are equal to 0 or 1, there must exist β such that equality holds in at least two distinct inequalities $\langle s, v(W_\beta)\rangle \geq c(W_\beta)$ associated to subspaces W_α among those which are used to define $\mathcal{P}_N$. We have already discarded the subspace $\{0\}$, so there must exist β such that W_β and V specify distinct inequalities. Thus $0 < \dim(W_\beta) < \dim(V)$. □

Proof of Theorem 5.1. Set $\mathcal{P} = \mathcal{P}(V, (V_j), (\phi_j))$. Consider first the base case $m = 1$. The datum is a pair of finite-dimensional vector spaces V, V_1 with a $\mathbb{Q}$–linear map $\phi\colon V \to V_1$. The polytope $\mathcal{P}$ is the set of all $s \in [0, 1]$ for which $s \dim(\phi(W)) \geq \dim(W)$ for every subspace $W \leq V$. If $\dim(V) = 0$ then $\mathcal{P}(V, (V_j), (\phi_j)) = [0, 1]$. If $\dim(V) > 0$ then since $\dim(\phi(W)) \leq \dim(W)$ for every subspace, the inequality can only hold if the nullspace of ϕ has dimension 0, and then only for $s = 1$. The nullspace of ϕ can be computed. So $\mathcal{P}$ can be computed when $m = 1$.

Suppose that $m \geq 2$. Let $(V, (V_j), (\phi_j))$ be given. Let $N = 0$. Recursively apply the following procedure.

Replace N by $N + 1$. Consider $\mathcal{P}_N$. Apply Lemma 5.4 to obtain a list of all extreme points τ of $\mathcal{P}_N$, and for each such τ which belongs to $(0, 1)^m$, a nonzero proper subspace $W(\tau) \leq V$ which is critical with respect to τ.

Examine each of these extreme points τ, to determine whether $\tau \in \mathcal{P}(V, (V_j), (\phi_j))$. There are three cases. Firstly, if $\tau \in (0, 1)^m$, then Proposition 5.5 may be invoked, using the critical subspace $W(\tau)$, to determine whether $\tau \in \mathcal{P}$.

Secondly, if some component τ_i of τ equals 1, let V' be the nullspace of ϕ_i. Set

$$\mathcal{P}' = \mathcal{P}(V', (V_j)_{j\neq i}, (\phi_j\big|_{V'})_{j\neq i}).$$

According to Lemma 5.6, $\tau \in \mathcal{P}$ if and only if $\widehat{\tau} = (\tau_j)_{j\neq i} \in \mathcal{P}'$. This polytope $\mathcal{P}'$ can be computed by the induction hypothesis, since the number of indices j has been reduced by one.

Finally, if some component τ_i of τ equals 0, then because the term $s_i \dim(\phi_i(W)) = 0$ contributes nothing to sums $\sum_{j=1}^m s_j \dim(\phi_j(W))$, $\tau \in \mathcal{P}$ if and only if $\widehat{\tau}$ belongs to $\mathcal{P}(V, (V_j)_{j\neq i}, (\phi_j)_{j\neq i})$. To determine whether $\widehat{\tau}$ belongs to this polytope requires again only an application of the induction hypothesis.

If every extreme point τ of $\mathcal{P}_N$ belongs to $\mathcal{P}$, then because $\mathcal{P}_N$ is the convex hull of its extreme points, $\mathcal{P}_N \subseteq \mathcal{P}$. The converse inclusion holds for every N, so in this case $\mathcal{P}_N = \mathcal{P}$. The algorithm halts, and returns the conclusion that $\mathcal{P} = \mathcal{P}_N$, along with information already computed: a list of the inequalities specified by all the subspaces $W_1, \ldots, W_N$, and a list of extreme points of $\mathcal{P}_N = \mathcal{P}$.

On the other hand, if at least one extreme point of $\mathcal{P}_N$ fails to belong to $\mathcal{P}$, then $\mathcal{P}_N \neq \mathcal{P}$. Then increment N by one, and repeat the above steps.

Lemma 5.7 guarantees that this procedure will halt after finitely many steps. □

5.2 On Computation of the Constraints Defining $\mathcal{P}$

In order to compute the set of inequalities (3.1), we would like to answer the following question: Given any group homomorphisms $\phi_1, \ldots, \phi_m$ and integers $0 \leq r, r_1, \ldots, r_m \leq d$, does there exist a subgroup $H \leq \mathbb{Z}^d$ such that

$$\operatorname{rank}(H) = r, \quad \operatorname{rank}(\phi_1(H)) = r_1, \quad \ldots, \quad \operatorname{rank}(\phi_m(H)) = r_m$$

or, in other words, is

$$r \leq \sum_{j=1}^{m} s_j \cdot r_j$$

one of the inequalities? Based on the following result, it is not known whether this problem is decidable for general $(\mathbb{Z}^d, (\mathbb{Z}^{d_j}), (\phi_j))$.

Theorem 5.9. *There exists an effective algorithm for computing the set of constraints* (3.1) *defining* $\mathcal{P}$ *if and only if there exists an effective algorithm to decide whether a system of polynomial equations with rational coefficients has a rational solution.*

The conclusion of Theorem 5.9 is equivalent to a positive answer to Hilbert's Tenth Problem for the rational numbers $\mathbb{Q}$ (see Definition 5.14), a longstanding open problem.

Let us fix some notation.

Notation 5.10 (see, e.g., [15]). *For a natural number d and ring R, we write $M_d(R)$ to denote the ring of d–by–d matrices with entries from R. (Note that elsewhere in this work we also use the notation $R^{m\times n}$ to denote the* set *of m–by–n matrices with entries from R.) We identify $M_d(R)$ with the endomorphism ring of the R–module R^d and thus may write elements of $M_d(R)$ as R–linear maps rather than as matrices. Via the usual coordinates, we may identify $M_d(R)$ with R^{d^2}. We write $R[x_1,\ldots,x_q]$ to denote the ring of polynomials over R in variables $x_1,\ldots,x_q$.*

Recall we are given $d, d_j \in \mathbb{N}$ and $\mathbb{Z}$–linear maps $\phi_j\colon \mathbb{Z}^d \to \mathbb{Z}^{d_j}$, for $j \in \{1,2,\ldots,m\}$ for some positive integer m. Without loss, we may assume each $d_j = d$, so each ϕ_j is an endomorphism of $\mathbb{Z}^d$. Each ϕ_j can also be interpreted as a $\mathbb{Q}$–linear map (from $\mathbb{Q}^d$ to $\mathbb{Q}^{d_j}$), represented by the same integer matrix.

Definition 5.11. *Given $m, d \in \mathbb{N}$, and a finite sequence $r, r_1,\ldots,r_m$ of natural numbers each bounded by d, we define the sets*

$$E_{d;r,r_1,\ldots,r_m} := \{(\phi_1,\ldots,\phi_m) \in (M_d(\mathbb{Z}))^m : (\exists H \le \mathbb{Z}^d)\ \mathrm{rank}(H) = r \textit{ and } \mathrm{rank}(\phi_j(H)) = r_j, 1 \le j \le m\},$$
$$E^{\mathbb{Q}}_{d;r,r_1,\ldots,r_m} := \{(\phi_1,\ldots,\phi_m) \in (M_d(\mathbb{Q}))^m : (\exists V \le \mathbb{Q}^d)\ \dim(V) = r \textit{ and } \dim(\phi_j(V)) = r_j, 1 \le j \le m\}.$$

Remark 5.12. *The question of whether a given m–tuple $(\phi_1,\ldots,\phi_m) \in (M_d(R))^m$ is a member of $E_{d;r,r_1,\ldots,r_m}$ (when $R = \mathbb{Z}$) or $E^{\mathbb{Q}}_{d;r,r_1,\ldots,r_m}$ (when $R = \mathbb{Q}$) is an instance of the problem of whether some system of polynomial equations has a solution over the ring R. We let B be a d–by–r matrix of variables, and construct a system of polynomial equations in the dr unknown entries of B and md^2 known entries of $\phi_1,\ldots,\phi_m$ that has a solution if and only if the aforementioned rank (or dimension) conditions are met. The condition* $\mathrm{rank}(M) = s$ *for a matrix M is equivalent to all $(s+1)$–by–$(s+1)$ minors of M equaling zero (i.e., the sum of their squares equaling zero), and at least one s–by–s minor being nonzero (i.e., the sum of their squares not equaling zero — see Remark 5.15). We construct two polynomial equations in this manner for $M = B$ (with $s = r$) and for each matrix $M = \phi_j B$ (with $s = r_j$).*

Lemma 5.13. *With the notation as in Definition 5.11, $E_{d;r,r_1,\ldots,r_m} = E^{\mathbb{Q}}_{d;r,r_1,\ldots,r_m} \cap (M_d(\mathbb{Z}))^m$.*

Proof. This result was already established in Lemma 3.14; we restate it here using the present notation. For the left-to-right inclusion, observe that if $H \le \mathbb{Z}^d$ witnesses that $(\phi_1,\ldots,\phi_m) \in E_{d;s,r_1,\ldots,r_m}$, then $H_{\mathbb{Q}}$ witnesses that $(\phi_1,\ldots,\phi_m) \in E^{\mathbb{Q}}_{d;r,r_1,\ldots,r_m}$. For the other inclusion, if $(\phi_1,\ldots,\phi_m) \in E^{\mathbb{Q}}_{d;r,r_1,\ldots,r_m} \cap (M_d(\mathbb{Z}))^m$ witnessed by $V \le \mathbb{Q}^d$, then we may find a subgroup $H = V \cap \mathbb{Z}^d$ of $\mathbb{Z}^d$, with $\mathrm{rank}(H) = \dim(V)$. Then $\dim(\phi_j(V)) = \mathrm{rank}(\phi_j(H)) = r_j$ showing that $(\phi_1,\ldots,\phi_m) \in E_{d;r,r_1,\ldots,r_m}$. □

Definition 5.14. Hilbert's Tenth Problem for $\mathbb{Q}$ *is the question of whether there is an algorithm which given a finite set of polynomials $f_1(x_1,\ldots,x_q),\ldots,f_p(x_1,\ldots,x_q) \in \mathbb{Q}[x_1,\ldots,x_q]$ (correctly) determines whether or not there is some $a \in \mathbb{Q}^q$ for which $f_1(a) = \cdots = f_p(a) = 0$.*

Remark 5.15. *One may modify the presentation of Hilbert's Tenth Problem for $\mathbb{Q}$ in various ways without affecting its truth value. For example, one may allow a condition of the form $g(a) \neq 0$ as this is equivalent to $(\exists b)(g(a)b - 1 = 0)$. On the other hand, using the fact that $x^2 + y^2 = 0 \iff x = 0 = y$, one may replace the finite sequence of polynomial equations with a single equality (see also Remark 5.18).*

Remark 5.16. *Hilbert's Tenth Problem, proper, asks for an algorithm to determine solvability in integers of finite systems of equations over $\mathbb{Z}$. From such an algorithm one could positively resolve Hilbert's Tenth Problem for $\mathbb{Q}$. However, by the celebrated theorem of Matiyasevich-Davis-Putnam-Robinson [17], no such algorithm exists. The problem for the rationals remains open. The most natural approach would be to reduce from the problem over $\mathbb{Q}$ to the problem over $\mathbb{Z}$, say, by showing that $\mathbb{Z}$ may be defined by an expression of the form*

$$a \in \mathbb{Z} \iff (\exists y_1)\cdots(\exists y_q)P(a; y_1,\ldots,y_q) = 0$$

for some fixed polynomial P. Königsmann [14] has shown that there is in fact a universal *definition of $\mathbb{Z}$ in $\mathbb{Q}$, that is, a formula of the form*

$$a \in \mathbb{Z} \iff (\forall y_1)\cdots(\forall y_q)\theta(a; y_1,\ldots,y_q) = 0$$

where θ is a finite Boolean combination of polynomial equations, but he also demonstrated that the existence of an existential definition of $\mathbb{Z}$ would violate the Bombieri-Lang conjecture. Königsmann's result shows that it is unlikely that Hilbert's Tenth Problem for $\mathbb{Q}$ can be resolved by reducing to the problem over $\mathbb{Z}$ using an existential definition of $\mathbb{Z}$ in $\mathbb{Q}$. However, it is conceivable that this problem could be resolved without such a definition.

Proof of Theorem 5.9 (necessity). Evidently, if Hilbert's Tenth Problem for $\mathbb{Q}$ has a positive solution, then there is an algorithm to (correctly) determine for $d \in \mathbb{N}$, $r, r_1, \ldots, r_m \leq d$ also in $\mathbb{N}$, and $(\phi_1, \ldots, \phi_m) \in (M_d(\mathbb{Z}))^m$ whether $(\phi_1, \ldots, \phi_m) \in E_{d;r,r_1,\ldots,r_m}$. By Lemma 5.13, $(\phi_1, \ldots, \phi_m) \in E_{d;r,r_1,\ldots,r_m}$ just in case $(\phi_1, \ldots, \phi_m) \in E^{\mathbb{Q}}_{d;r,r_1,\ldots,r_m}$. This last condition leads to an instance of Hilbert's Tenth Problem (for $\mathbb{Q}$) for the set of rational polynomial equations given in Remark 5.12. □

Notation 5.17. *Given a set $S \subseteq \mathbb{Q}[x_1, \ldots, x_q]$ of polynomials, we denote the set of rational solutions to the equations $f = 0$ as f ranges through S by*

$$V(S)(\mathbb{Q}) := \{a \in \mathbb{Q}^q : (\forall f \in S) f(a) = 0\}.$$

Remark 5.18. *Since $V(S)(\mathbb{Q})$ is an algebraic set over a field, Hilbert's basis theorem shows that we may always take S to be finite. Then by replacing S with $S' := \{\sum_{f \in S} f^2\}$, one sees that S may be assumed to consist of a single polynomial. While the reduction to finite S is relevant to our argument, the reduction to a single equation is not.*

Definition 5.19. *For any natural number t, we say that the set $D \subseteq \mathbb{Q}^t$ is* Diophantine *if there is some $q - t \in \mathbb{N}$ and a set $S \subseteq \mathbb{Q}[x_1, \ldots, x_t; y_1, \ldots, y_{q-t}]$ for which*

$$D = \{a \in \mathbb{Q}^t : (\exists b \in \mathbb{Q}^{q-t})(a; b) \in V(S)(\mathbb{Q})\}.$$

We will show sufficiency in Theorem 5.9 by establishing a stronger result: namely, that an algorithm to decide membership in sets of the form $E_{d;r,r_1,\ldots,r_m}$ could also be used to decide membership in any Diophantine set. (Hilbert's Tenth Problem for $\mathbb{Q}$ concerns membership in the specific Diophantine set $V(S)(\mathbb{Q})$.)

With the next lemma, we use a standard trick of replacing composite terms with single applications of the basic operations to put a general Diophantine set in a standard form (see, e.g., [24]).

Lemma 5.20. *Given any finite set of polynomials $S \subset \mathbb{Q}[x_1, \ldots, x_q]$, let $d := \max_{f \in S} \max_{i=1}^q \deg_{x_i}(f)$ and $\mathcal{D} := \{0, 1, \ldots, d\}^q$. There is another set of variables $\{u_\alpha\}_{\alpha \in \mathcal{D}}$ and another finite set $S' \subset \mathbb{Q}[\{u_\alpha\}_{\alpha \in \mathcal{D}}]$ consisting entirely of affine polynomials (polynomials of the form $c + \sum c_\alpha u_\alpha$ where not all c and c_α are zero) and polynomials of the form $u_\alpha u_\beta - u_\gamma$ with α, β, and γ distinct, so that $V(S)(\mathbb{Q}) = \pi(V(S')(\mathbb{Q}))$ where $\pi \colon \mathbb{Q}^{\mathcal{D}} \to \mathbb{Q}^q$ is given by*

$$(u_\alpha)_{\alpha \in \mathcal{D}} \mapsto (u_{(1,0,\ldots,0)}, u_{(0,1,0,\ldots,0)}, \ldots, u_{(0,\ldots,0,1)}).$$

Proof. Let $T \subset \mathbb{Q}[\{u_\alpha\}_{\alpha \in \mathcal{D}}]$ consist of

- $u_{(0,\ldots,0)} - 1$ and
- $u_{\alpha+\beta} - u_\alpha u_\beta$ for $(\alpha + \beta) \in \mathcal{D}$, $\alpha \neq \mathbf{0}$ and $\beta \neq \mathbf{0}$.

Define $\chi \colon \mathbb{Q}^q \to \mathbb{Q}^{\mathcal{D}}$ by

$$(x_1, \ldots, x_q) \mapsto (x^\alpha)_{\alpha \in \mathcal{D}}$$

where $x^\alpha := \prod_{j=1}^q x_j^{\alpha_j}$. One sees immediately that χ induces a bijection $\chi \colon \mathbb{Q}^q \to V(T)(\mathbb{Q})$ with inverse $\pi\big|_{V(T)(\mathbb{Q})}$.

Let S' be the set containing T and the polynomials $\sum_{\alpha \in \mathcal{D}} c_\alpha u_\alpha$ for which $\sum_{\alpha \in \mathcal{D}} c_\alpha x^\alpha \in S$.

One checks that if $a \in \mathbb{Q}^q$, then $\chi(a) \in V(S')(\mathbb{Q})$ if and only if $a \in V(S)(\mathbb{Q})$. Applying π, and noting that π is the inverse to χ on $V(T)(\mathbb{Q})$, the result follows. □

Notation 5.21. *For the remainder of this argument, we call a set enjoying the properties identified for S' (namely that each polynomial is either affine or of the form $u_\alpha u_\beta - u_\gamma$) a* basic set.

Proof of Theorem 5.9 (sufficiency). It follows from Remark 5.18 and Lemma 5.20 that the membership problem for a general Diophantine set may be reduced to the membership problem for a Diophantine set defined by a finite basic set of equations. Let $S \subseteq \mathbb{Q}[x_1, \ldots, x_q]$ be a finite basic set of equations and let $t \leq q$ be some natural number, We now show that there are natural numbers $\mu, \nu, \rho, \rho_1, \ldots, \rho_\mu$ and a computable function $f \colon \mathbb{Q}^t \to (M_\nu(\mathbb{Z}))^\mu$ so that for $a \in \mathbb{Q}^t$ one has that there is some $b \in \mathbb{Q}^{q-t}$ with $(a, b) \in V(S)(\mathbb{Q})$ if and only if $f(a) \in E_{\nu;\rho,\rho_1,\ldots,\rho_\mu}$.

List the ℓ affine polynomials in S as

$$\lambda_{0,1} + \sum_{i=1}^q \lambda_{i,1} x_i, \qquad \ldots, \qquad \lambda_{0,\ell} + \sum_{i=1}^q \lambda_{i,\ell} x_i$$

and the k polynomials expressing multiplicative relations in S as

$$x_{i_{1,1}} x_{i_{2,1}} - x_{i_{3,1}}, \qquad \ldots, \qquad x_{i_{1,k}} x_{i_{2,k}} - x_{i_{3,k}}.$$

Note that by scaling, we may assume that all of the coefficients $\lambda_{i,j}$ are integers.

We shall take $\mu := 4 + q + t + |S|$, $\nu := 2q + 2$, $\rho := 2$ and the sequence $\rho_1, \ldots, \rho_\mu$ to consist of $4 + q + k$ ones followed by $t + \ell$ zeros. Let us describe the map $f \colon \mathbb{Q}^t \to (M_\nu(\mathbb{Z}))^\mu$ by expressing each coordinate. For the sake of notation, our coordinates on $\mathbb{Q}^\nu$ are $(u; v) := (u_0, u_1 \ldots, u_q; v_0, v_1, \ldots, v_q)$.

A. The map f_1 is constant taking the value $(u; v) \mapsto (u_0, 0, \ldots, 0)$.

B. The map f_2 is constant taking the value $(u; v) \mapsto (v_0, 0, \ldots, 0)$.

C. The map f_3 is constant taking the value $(u; v) \mapsto (u_0, \ldots, u_q; 0, \ldots, 0)$.

D. The map f_4 is constant taking the value $(u; v) \mapsto (v_0, \ldots, v_q; 0, \ldots, 0)$.

E. The map f_{4+j} (for $0 < j \leq q$) is constant taking the value $(u; v) \mapsto (u_0 - v_0, u_j - v_j, 0, \ldots, 0)$.

F. The map f_{4+q+j} (for $0 < j \leq k$) is constant taking the value $(u; v) \mapsto (u_{i_{1,j}} + v_0, u_{i_{3,j}} + v_{i_{2,j}}, 0, \ldots, 0)$.

G. The map $f_{4+q+k+j}$ (for $0 < j \leq t$) takes $a = (\frac{p_1}{q_1}, \ldots, \frac{p_t}{q_t})$ (written in lowest terms) to the linear map $(u; v) \mapsto (p_j u_0 - q_j u_j, 0, \ldots, 0)$.

H. The map $f_{4+q+k+t+j}$ (for $0 < j \leq \ell$) takes the value $(u; v) \mapsto (\sum_{i=0}^{q} \lambda_{i,j} u_i, 0, \ldots, 0)$.

Note that only the components $f_{4+q+k+j}$ for $0 < j \leq t$ actually depend on $(a_1, \ldots, a_t) \in \mathbb{Q}^t$.

Let us check that this construction works. First, suppose that $a \in \mathbb{Q}^t$ and that there is some $b \in \mathbb{Q}^{q-t}$ for which $(a, b) \in V(S)(\mathbb{Q})$. For the sake of notation, we write $c = (c_1, \ldots, c_q) := (a_1, \ldots, a_t, b_1, \ldots, b_{q-t})$.

Let $V := \mathbb{Q}(1, c_1, \ldots, c_q, 0, \ldots, 0) + \mathbb{Q}(0, \ldots, 0, 1, c_1, \ldots, c_q)$. We will check now that V witnesses that $f(a) \in E_{\nu,\rho,\rho_1,\ldots,\rho_\mu}$. Note that a general element of V takes the form $(\alpha, \alpha c_1, \ldots, \alpha c_q, \beta, \beta c_1, \ldots, \beta c_q)$ for $(\alpha, \beta) \in \mathbb{Q}^2$. Throughout the rest of this proof, when we speak of a general element of some image of V, we shall write α and β as variables over $\mathbb{Q}$.

Visibly, $\dim(V) = 2 = \rho$.

Clearly, $f_1(a)(V) = \mathbb{Q}(1, 0, \ldots, 0) = f_2(a)(V)$, so that $\rho_1 = \dim(f_1(a)(V)) = 1 = \dim(f_2(a)(V)) = \rho_2$ as required.

Likewise, $f_3(a)(V) = \mathbb{Q}(1, c_1, \ldots, c_q, 0, \ldots, 0) = f_4(a)(V)$, so that $\rho_3 = \dim(f_3(a)(V)) = 1 = \dim(f_4(a)(V)) = \rho_4$.

For $0 < j \leq q$ the general element of $f_{4+j}(a)(V)$ has the form $(\alpha - \beta, \alpha c_j - \beta c_j, 0, \ldots, 0) = (\alpha - \beta)(1, c_j, 0, \ldots, 0)$. Thus, $f_{4+j}(a)(V) = \mathbb{Q}(1, c_j, 0, \ldots, 0)$ has dimension $\rho_{4+j} = 1$.

For $0 < j \leq k$ we have $c_{i_{3,j}} = c_{i_{1,j}} c_{i_{2,j}}$, the general element of $f_{4+q+j}(a)(V)$ has the form $(\alpha c_{i_{1,j}} + \beta, \alpha c_{i_{3,j}} + \beta c_{i_{2,j}}, 0, \ldots, 0) = (\alpha c_{i_{1,j}} + \beta, \alpha c_{i_{1,j}} c_{i_{2,j}} + \beta c_{i_{2,j}}, 0, \ldots, 0) = (\alpha c_{i_{1,j}} + \beta)(1, c_{i_{2,j}}, 0, \ldots, 0)$ so we have that $\rho_{4+q+j} = \dim(f_{4+q+j}(a)(V)) = 1$.

For $0 < j \leq t$, the general element of $f_{4+q+k+j}(a)(V)$ has the form $(p_j \alpha - q_j \alpha a_j, 0, \ldots, 0) = \mathbf{0}$. That is, $\rho_{4+q+k+j} = \dim(f_{4+q+k+j}(a)(V)) = 0$.

Finally, if $0 < j \leq \ell$, then the general element of $f_{4+q+k+t+j}(a)(V)$ has the form $(\lambda_{0,j}\alpha + \sum_{i=1}^{q} \lambda_{i,j} \alpha c_i, 0, \ldots, 0) = \mathbf{0}$ since $\lambda_{0,j} + \sum_{i=1}^{q} \lambda_{i,j} c_i = 0$. So $\rho_{4+q+k+t+j} = \dim(f_{4+q+k+t+j}(a)(V)) = 0$.

Thus, we have verified that if $a \in \mathbb{Q}^t$ and there is some $b \in \mathbb{Q}^{d-t}$ with $(a, b) \in V(S)(\mathbb{Q})$, then $f(a) \in E_{\nu;\rho,\rho_1,\ldots,\rho_\mu}$.

Conversely, suppose that $a = (\frac{p_1}{q_1}, \ldots, \frac{p_t}{q_t}) \in \mathbb{Q}^t$ and that $f(a) \in E_{\nu;\rho,\rho_1,\ldots,\rho_\mu}$, with the same $\mu, \nu, \rho, \rho_1, \ldots, \rho_\mu$. Let $V \leq \mathbb{Q}^\nu$ have $\dim(V) = \rho$ witnessing that $f(a) \in E_{\nu;\rho,\rho_1,\ldots,\rho_\mu}$.

Lemma 5.22. *There are elements g and h in V for which $g = (g_0, \ldots, g_q; 0, \ldots, 0)$, $h = (0, \ldots, 0; h_0, \ldots, h_q)$, $g_0 = h_0 = 1$, and $\mathbb{Q}g + \mathbb{Q}h = V$.*

Proof. The following is an implementation of row reduction. Let the elements $d = (d_0, \ldots, d_q; d'_0, \ldots, d'_q)$ and $e = (e_0, \ldots, e_q; e'_0, \ldots, e'_q)$ be a basis for V. Since $\dim(f_1(a)(V)) = 1$, at the cost of reversing d and e and multiplying by a scalar, we may assume that $d_0 = 1$. Since $\dim(f_3(a)(V)) = 1$, we may find a scalar γ for which $(\gamma d_0, \ldots, \gamma d_q) = (e_0, \ldots, e_q)$. Set $\widetilde{g} := e - \gamma d$. Write $\widetilde{g} = (0, \ldots, 0, \widetilde{g}_0, \ldots, \widetilde{g}_q)$. Since $\widetilde{g}$ is linearly independent from e and $\dim(f_4(a)(V)) = 1$, we see that there is some scalar δ for which $(\delta\widetilde{g}_0, \ldots, \delta\widetilde{g}_q) = (d'_0, \ldots, d'_q)$. Set $h := d - \delta\widetilde{g}$. Using the fact that $\dim(f_2(a)(V)) = 1$ we see that $\widetilde{g}_0 \neq 0$. Set $g := \widetilde{g}_0^{-1}\widetilde{g}$. □

Lemma 5.23. *For $0 \le j \le q$ we have $g_j = h_j$.*

Proof. We arranged $g_0 = 1 = h_0$ in Lemma 5.22. The general element of V has the form $\alpha h + \beta g$ for some $(\alpha, \beta) \in \mathbb{Q}^2$. For $0 < j \le q$, the general element of $f_{4+j}(a)(V)$ has the form $(\alpha h_0 - \beta h_0, \alpha h_j - \beta g_j, 0, \ldots, 0) = ((\alpha - \beta)h_0, (\alpha - \beta)h_j + \beta(h_j - g_j), 0, \ldots, 0)$. Since $h_0 \ne 0$, if $h_j \ne g_j$, then this vector space would have dimension two, contrary to the requirement that $f(a) \in E_{\nu;\rho,\rho_1,\ldots,\rho_\mu}$. □

Lemma 5.24. *For $0 < j \le t$ we have $h_j = a_j$.*

Proof. The image of $\alpha h + \beta g$ under $f_{4+q+k+j}(a)$ is $(p_j\alpha - q_j\alpha h_j, 0, \ldots, 0)$ where $a_j = \frac{p_j}{q_j}$ in lowest terms. Since $\dim(f_{4+q+k+j}(a)(V)) = 0$, we have $q_j h_j = p_j$. That is, $h_j = a_j$. □

Lemma 5.25. *For any $F \in S$, we have $F(h_1, \ldots, h_q) = 0$.*

Proof. If F is an affine polynomial, that is, if $F = \lambda_{0,j} + \sum_{i=1}^q \lambda_{i,j} x_i$ for some $0 < j \le \ell$, then because $\dim(f_{4+q+k+t+j}(a)(V)) = 0$, we have $\lambda_{0,j} + \sum_{i=1}^q \lambda_{i,j} h_i = 0$. On the other hand, if F is a multiplicative relation, that is, if $F = x_{i_{1,j}} x_{i_{2,j}} - x_{i_{3,j}}$ for some $0 < j \le k$, then because $\dim(f_{4+q+j}(a)(V)) = 1$ we see that there is some scalar γ so that for any pair (α, β) we have $\gamma(\alpha h_{i_{1,j}} + \beta) = \alpha h_{i_{3,j}} + \beta h_{i_{2,j}}$. Specializing to $\beta = 0$ and $\alpha = 1$, we have $\gamma = \frac{h_{i_{3,j}}}{h_{i_{1,j}}}$ (unless both are zero, in which case the equality $h_{i_{1,j}} h_{i_{2,j}} = h_{i_{3,j}}$ holds anyway), which we substitute to obtain $\frac{h_{i_{3,j}}}{h_{i_{1,j}}} = h_{i_{2,j}}$, or $h_{i_{2,j}} h_{i_{1,j}} = h_{i_{3,j}}$. □

Taking $b := (h_{t+1}, \ldots, h_q)$ we see that $(a, b) \in V(S)(\mathbb{Q})$. □

5.3 Computing upper and lower bounds on s_{HBL}

5.3.1 Bounding s_{HBL} below: Approximating $\mathcal{P}$ by a superpolytope via sampling constraints

While the approach in Section 5.1 demonstrates that $\mathcal{P}$ can be computed exactly, it follows from (5.1) that we can terminate the algorithm at any step N and obtain a superpolytope $\mathcal{P}_N \supseteq \mathcal{P}$. Thus, the minimum sum $s_{\mathrm{HBL},N}$ of $s \in \mathcal{P}_N$ is a lower bound on the desired s_{HBL}. Since our communication lower bound is inversely proportional to $M^{s_{\mathrm{HBL}}-1}$, $s_{\mathrm{HBL},N}$ may lead to an erroneous bound which is larger than the true lower bound. But the erroneous bound may still be attainable, possibly leading to a faster algorithm.

We note that, since $s_{\mathrm{HBL},N}$ is nondecreasing in N, the approximation can only improve as we take more steps of the algorithm. The question remains of how to systematically choose a sequence of subspaces $W_1, \ldots, W_N$ of V that are likely to yield the best estimate of s_{HBL}, i.e., the fastest growing $s_{\mathrm{HBL},N}$. Furthermore, even if $\mathcal{P}_N$ is a proper superpolytope, it may still be that $s_{\mathrm{HBL},N} = s_{\mathrm{HBL}}$, and we could stop the algorithm earlier. In Part 2 of this work, we will discuss ways to improve the efficiency of the approach in Section 5.1, including choosing the subspaces W_i and detecting this early-termination condition (see Section 8).

5.3.2 Bounding s_{HBL} above: Approximating $\mathcal{P}$ by a subpolytope via embedding into $\mathbb{R}^d$

The approximation just presented in Section 5.3.1 yielded an underestimate of s_{HBL}. Now we discuss a different approximation that yields an overestimate of s_{HBL}. If the overestimate and underestimate agree, then we have a proof of optimality, and if they are close, it is also useful.

Recall in Section 5.2, we hoped to answer the following (possibly undecidable) question.

"Given integers $0 \le r, r_1, \ldots, r_m \le d$, is there a subgroup $H \le \mathbb{Z}^d$ such that $\mathrm{rank}(H) = r$, $\mathrm{rank}(\phi_1(H)) = r_1, \ldots, \mathrm{rank}(\phi_m(H)) = r_m$?"

If such an H exists, we know the linear constraint $r \le \sum_{j=1}^m s_j r_j$ is one of the conditions (3.1), which define the polytope $\mathcal{P}$ of feasible solutions $(s_1, \ldots, s_m)$. We now ask the related question,

"...is there a subspace $V \le \mathbb{R}^d$ such that $\dim(V) = r$, $\dim(\phi_1(V)) = r_1, \ldots, \dim(\phi_m(V)) = r_m$?"

Here, we reinterpret ϕ_j as an $\mathbb{R}$–linear map. It turns out that computing the set of inequalities (3.8) for the HBL datum $(\mathbb{R}^d, (\mathbb{R}^{d_j}), (\phi_j))$ is easier than in the previous case, in the sense of being Tarski-decidable. Following the notation in Section 5.2, we define

$$E^{\mathbb{R}}_{d;r,r_1,\ldots,r_m} := \{(\phi_1, \ldots, \phi_m) \in (M_d(\mathbb{R}))^m : (\exists V \le \mathbb{R}^d)\ \dim(V) = r \text{ and } \dim(\phi_j(V)) = r_j, 1 \le j \le m\}.$$

Theorem 5.26. *It is Tarski-decidable [21] to decide membership in the sets* $E^{\mathbb{R}}_{d;r,r_1,\ldots,r_m}$.

Proof. There is an effective algorithm (the cylindrical algebraic decomposition [7]) to decide whether a system of real polynomial equations (e.g., that in Remark 5.12) has a real solution. □

As opposed to the rational embedding considered in Section 5.2, this real embedding is a *relaxation* of the original question. That is, if $H \leq \mathbb{Z}^d$ exists, then V, the real vector subspace of $\mathbb{R}^d$ generated by H, exists with $\dim(V) = \operatorname{rank}(H)$; however, there may be V which do not correspond to any $H \leq \mathbb{Z}^d$. In other words, the existence of V is a necessary but not sufficient condition for the existence of such a subgroup $H \leq \mathbb{Z}^d$. Since $\dim(\phi_j(V)) = \operatorname{rank}(\phi_j(H))$, we see that the set of inequalities (3.8) for the vector space HBL datum $(\mathbb{R}^d, (\mathbb{R}^{d_j}), (\phi_j))$ is a superset of the inequalities (3.1) for the Abelian group HBL datum $(\mathbb{Z}^d, (\mathbb{Z}^{d_j}), (\phi_j))$ (or equivalently, for the rational embedding considered in Section 5.2). In terms of the polytopes defined by these inequalities, we have

$$\bar{\mathcal{P}} := \mathcal{P}(\mathbb{R}^d, (\mathbb{R}^{d_j}), (\phi_j)) \subseteq \mathcal{P},$$

i.e., the constraints for the relaxed problem give a polytope $\bar{\mathcal{P}}$ of feasible solutions $s = (s_1, \ldots, s_m)$, which is a subpolytope (subset) of $\mathcal{P}$. If the relaxed constraints are a proper superset, then $\bar{\mathcal{P}}$ is a proper subpolytope. This means that the conclusion (3.2) of Theorem 3.2 is still valid for any $s \in \bar{\mathcal{P}}$, but the upper bound may be too large. Therefore, the communication lower bound that is inversely proportional to $M^{\bar{s}_{\text{HBL}}-1}$, where $\bar{s}_{\text{HBL}} := \min_{s\in\bar{\mathcal{P}}} \sum_{j=1}^m s_j$, is also still valid. However, it may be smaller than the lower bound that is inversely proportional to $M^{s_{\text{HBL}}-1}$, where $s_{\text{HBL}} := \min_{s\in\mathcal{P}} \sum_{j=1}^m s_j$.

In other words, it is Tarski-decidable to write down a communication lower bound, but it may be strictly smaller than the best lower bound implied by Theorem 3.2, and obtained by the approach in Section 5.1.

6 Easily computable communication lower bounds

As mentioned in Section 5, there may be many *equivalent* linear programs to determine s_{HBL} that may not include the set of inequalities (3.1). The approach in Section 5.1 shows that it is always possible to write down one such linear program. In this section, we discuss three practical cases where we can write down another (equivalent) linear program without applying the approach in Section 5.1.

First, in Section 6.1, we demonstrate some transformations in that let us simplify the input HBL datum $(\mathbb{Z}^d, (\mathbb{Z}^{d_j}), (\phi_j))$ without altering s_{HBL}. Then in Section 6.2 we examine two trivial cases where we can immediately write down an equivalent linear program. In the first case (Section 6.2.1), we show that we can always immediately recognize when the feasible region is empty; therefore no s_{HBL} exists and arbitrary data reuse is possible. In the second case (Section 6.2.2), when one or more ϕ_j is an injection, we can immediately determine that $s_{\text{HBL}} = 1$, that is, no asymptotic data reuse is possible. Lastly, in Section 6.3, we present the most general case solved explicitly in this paper, when the subscripts of each array are a subset of the loop indices. We apply this to several examples: the direct N–body problem, database join, matrix-vector multiplication, and tensor contractions, as well as three examples seen earlier: matrix-matrix multiplication, "complicated code" (from Section 1), and computing matrix powers (from Section 4).

There are variety of other situations in which we can effectively compute the communication lower bound, without applying the approach in Section 5.1. These are summarized in Section 8, and will appear in detail in Part 2 of this paper.

6.1 Simplifying the linear constraints of (3.1)

If $\operatorname{rank}(\phi_j(\mathbb{Z}^d)) = 0$ (e.g., array A_j is a scalar), we say subscript ϕ_j is a trivial map, since it maps all indices $\mathcal{I}$ to the same subscript. The following remark shows that we can ignore trivial maps when computing a communication lower bound, without affecting s_{HBL}.

Remark 6.1. *Suppose we are given the HBL datum* $(\mathbb{Z}^d, (\mathbb{Z}^{d_j}), (\phi_j))$ *which satisfies* $\operatorname{rank}(\phi_j(\mathbb{Z}^d)) = 0$ *for indices* $k < j \leq m$. *Then* (3.1) *becomes*

$$\operatorname{rank}(H) \leq \sum_{j=1}^{k} s_j \operatorname{rank}(\phi_j(H)) \qquad \textit{for all subgroups } H \leq \mathbb{Z}^d.$$

So given feasible exponents $(s_j)_{j=1}^k$ *for the HBL datum* $(\mathbb{Z}^d, (\mathbb{Z}^{d_j})_{j=1}^k, (\phi_j)_{j=1}^k)$, *we are free to pick* $s_j \in [0,1]$ *for* $k < j \leq m$. *Towards minimizing* s_{HBL}, *we would, of course, pick* $s_j = 0$ *for these indices. So, we ignore trivial maps when computing a communication lower bound.*

It is also possible that A_i and A_j have the same subscript $\phi_i = \phi_j$. The following remark shows that, as with trivial maps, we can ignore such duplicate subscripts, without affecting s_{HBL}.

Remark 6.2. *Given the HBL datum $(\mathbb{Z}^d, (\mathbb{Z}^{d_j}), (\phi_j))$, suppose* $\ker(\phi_k) = \cdots = \ker(\phi_m)$*. Then $\phi_i(H) \cong \phi_j(H)$ for any $k \le i, j \le m$. Then the constraint* (3.1) *becomes*

$$\operatorname{rank}(H) \le \sum_{j=1}^{k-1} s_j \operatorname{rank}(\phi_j(H)) + \operatorname{rank}(\phi_k(H)) \sum_{j=k}^{m} s_j = \sum_{j=1}^{k} t_j \operatorname{rank}(\phi_j(H)) \qquad \textit{for all subgroups } H \le \mathbb{Z}^d.$$

So given feasible exponents $t = (t_j)_{j=1}^k$ for the HBL datum $(\mathbb{Z}^d, (\mathbb{Z}^{d_j})_{j=1}^k, (\phi_j)_{j=1}^k)$, we have a family of feasible exponents $\{s = (s_j)_{j=1}^m \in [0,1]^m\}$ for the original datum, satisfying $s_j = t_j$ for $1 \le j < k$ and $s_k + \cdots + s_m = t_k$. Our objective s_{HBL} only depends on the sum of the exponents (i.e., computing t suffices); thus we ignore duplicate maps when computing a communication lower bound.

So in the remainder of this section, we assume that the group homomorphisms $\phi_1, \ldots, \phi_m$ are distinct and nontrivial. The arrays A_j however, may overlap in memory addresses; this possibility does not affect our asymptotic lower bound, given our assumption from Section 4 that m is a constant, negligible compared to M.

6.2 Trivial Cases

There are a few trivial cases where no work is required to find the communication lower bound: when the homomorphisms' kernels have a nontrivial intersection, and when at least one homomorphism is an injection. These cover the cases where $d = 1$ or $m = 1$. (When $m = 0$ there are no arrays, and when $d = 0$ there are no loops, cases we ignore.)

The following lemma shows that $s_{\text{HBL}} = 0$ only arises in the trivial $d = 0$ case; otherwise, when the constraints (3.1) are feasible and s_{HBL} exists, then $s_{\text{HBL}} \ge 1$.

Lemma 6.3. *If $d > 0$ and $s \in [0,1]^m$ satisfies* (3.1)*, then $\sum_{j=1}^m s_j \ge 1$.*

Proof. If not, then any nontrivial subgroup is supercritical with respect to s. □

6.2.1 Infeasible Case

The following result, a companion to Theorem 3.2, gives a simpler necessary and sufficient condition for there to exist some exponent s which satisfies (3.2).

Lemma 6.4. *There exists $s \in [0,1]^m$ for which* (3.2) *holds if and only if*

$$\bigcap_{j=1}^{m} \ker(\phi_j) = \{0\}. \tag{6.1}$$

Proof. If $\bigcap_j \ker(\phi_j) = \{0\}$, then Lemma 3.27 asserts that (3.2) holds with all s_j equal to 1. Conversely, if $H = \bigcap_j \ker(\phi_j)$ is nontrivial, then for any finite nonempty subset $E \subset H$, $\phi_j(E) = \{0\}$ for every index j, so $|\phi_j(E)| = 1$. Thus the inequality (3.2) fails to hold for any $E \subset H$ of cardinality ≥ 2. □

So, if $\bigcap_j \ker(\phi_j) \ne \{0\}$, then there is no s_{HBL}, and there are arbitrarily large sets E that access at most $m \ll M$ operands; if operands may reside in cache before/after the program's execution, then a communication lower bound of 0 is attainable (see Section 7.1.1).

If $m = 1$ or $d = 1$, then either we fall into this case, or that of Section 6.2.2.

Example: Matrix Powering (Part 2/4). Recall the 'Original Code' for computing $B = A^k$ (for odd k) from part 1/4 of this example. The lines $C = A \cdot B$ and $B = A \cdot C$ are matrix multiplications, i.e., each is 3 nested loops (over i_2, i_3, i_4). As discussed in part 1/4, we ignore the second matrix multiplication, leaving only $C = A \cdot B$. This yields $\phi_A(i_1, i_2, i_3, i_4) = (i_2, i_4)$, $\phi_B(i_1, i_2, i_3, i_4) = (i_3, i_4)$, and $\phi_C(i_1, i_2, i_3, i_4) = (i_2, i_3)$. Observe that the subgroup $\ker(\phi_A) \cap \ker(\phi_B) \cap \ker(\phi_C) = \langle i_2 = i_3 = i_4 = 0 \rangle < \mathbb{Z}^4$ is nontrivial, so by Lemma 6.4, $\mathcal{P}$ is empty and no s_{HBL} exists. This tells us that there is no lower bound on communication that is proportional to the number of loop iterations $\lfloor k/2 \rfloor N^3$. Indeed, if one considers an execution where the i_1 loop is made the innermost, then we can increase k arbitrarily without additional communication. (Recall that our bounds ignore data dependencies, and do not require correctness.) So to derive a lower bound and corresponding optimal algorithm, we apply the approach of *imposing Reads and Writes* from Section 4. We will continue this example in Section 6.3, after introducing Theorem 6.6. △

6.2.2 Injective Case

If one of the array references is an injection, then each loop iteration will require (at least) one unique operand, so at most M iterations are possible with M operands in cache. This observation is confirmed by the following lemma.

Lemma 6.5. *If ϕ_k is injective, then the minimal $s_{\mathrm{HBL}} = 1$.*

Proof. We see that $s \coloneqq e_k \in \mathcal{P}$ by rewriting (3.1) as

$$1 \leq s_k + \sum_{j \neq k} s_j \operatorname{rank}(\phi_j(H)) / \operatorname{rank}(H) \qquad \text{for all subgroups } H \leq \mathbb{Z}^d.$$

The sum $s_{\mathrm{HBL}} = 1$ is minimal due to Lemma 6.3. □

As anticipated, the argument in Section 4 gives a lower bound $\Omega(M \cdot |\mathcal{Z}|/F) = \Omega(|\mathcal{Z}|)$, that is, no (asymptotic) data reuse is possible.

Example: Matrix-Vector Multiplication (Part 1/3). The code is

$$\begin{aligned} &\textbf{for } i_1 = 1 : N, \quad \textbf{for } i_2 = 1 : N, \\ &\qquad y(i_1) \mathrel{+}= A(i_1, i_2) \cdot x(i_2) \end{aligned}$$

We get $\phi_y(i_1, i_2) = (i_1)$, $\phi_A(i_1, i_2) = (i_1, i_2)$, and $\phi_x(i_1, i_2) = (i_2)$. Clearly, $\operatorname{rank}(\phi_A(\mathbb{Z}^2)) = 2$, i.e., ϕ_A is an injection, so by Lemma 6.5 we have $s_{\mathrm{HBL}} = 1$. Each iteration $\mathcal{I} = (i_1, i_2)$ requires a unique entry $A(i_1, i_2)$, so at most $M^{s_{\mathrm{HBL}}} = M$ iterations are possible with M operands. In part 2/3 of this example (in Section 6.3, below) we arrive at the lower bound $s_{\mathrm{HBL}} = 1$ in a different manner (via Theorem 6.6). △

6.3 Product Case

Recall the matrix multiplication example, which iterates over $\mathbb{Z}^3$ indexed by three loop indices (i_1, i_2, i_3), with inner loop $C(i_1, i_2) \mathrel{+}= A(i_1, i_3) \cdot B(i_3, i_2)$, which gives rise to 3 homomorphisms

$$\phi_A(i_1, i_2, i_3) = (i_1, i_3), \quad \phi_B(i_1, i_2, i_3) = (i_3, i_2), \quad \phi_C(i_1, i_2, i_3) = (i_1, j_2),$$

all of which simply choose subsets of the loop indices i_1, i_2, i_3. Similarly, for other linear algebra algorithms, tensor contractions, direct N–body simulations, and other examples discussed below, the ϕ_j simply choose subsets of loop indices. In this case, one can straightforwardly write down a simple linear program for the optimal exponents of Theorem 3.2. We present this result as Theorem 6.6 below (this is a special case, with a simpler proof, of the more general result in [6, Proposition 7.1]). Using Theorem 6.6 we also give a number of concrete examples, some known (e.g., linear algebra) and some new.

We use the following notation. $\mathbb{Z}^d$ will be our d–dimensional set including all possible tuples of loop indices $\mathcal{I} = (i_1, \ldots, i_d)$. The homomorphisms $\phi_1, \ldots, \phi_m$ all choose subsets of these indices $S_1, \ldots, S_m$, i.e., each $S_j \subseteq \{1, \ldots, d\}$ indicates which indices ϕ_j chooses. If an array reference uses multiple copies of a subscript, e.g., $A_3(i_1, i_1, i_2)$, then the corresponding $S_3 = \{1, 2\}$ has just one copy of each subscript.

Theorem 6.6. *Suppose each ϕ_j chooses a subset S_j of the loop indices, as described above. Let $H_i \leq \mathbb{Z}^d$ denote the subgroup $H_i = \langle e_i \rangle$, i.e., nonzero only in the i^{th} coordinate, for $i \in \{1, \ldots, d\}$. Then, for any $s_j \in [0, 1]$,*

$$\operatorname{rank}(H_i) \leq \sum_{j=1}^{m} s_j \operatorname{rank}(\phi_j(H_i)) \qquad \textit{for all } i \in \{1, 2, \ldots, d\} \tag{6.2}$$

if and only if

$$\operatorname{rank}(H) \leq \sum_{j=1}^{m} s_j \operatorname{rank}(\phi_j(H)) \qquad \textit{for all subgroups } H \leq \mathbb{Z}^d. \tag{3.1}$$

Proof. Necessity follows from the fact that the constraints (6.2) are a subset of (3.1).

To show sufficiency, we rewrite hypothesis (6.2) as

$$\operatorname{rank}(H_i) = 1 \leq \sum_{j=1}^{m} s_j \delta(j, i) \qquad \text{for all } i \in \{1, 2, \ldots, d\},$$

where $\delta(j,i)=1$ if $i \in S_j$, and $\delta(j,i)=0$ otherwise. Let $H \leq \mathbb{Z}^d$ have $\text{rank}(H)=h$. Express H as the set spanned by integer linear combinations of columns of some rank h integer matrix M_H of dimension d–by–h. Pick some full rank h–by–h submatrix M' of M_H. Suppose M' lies in rows of M_H in the set $\mathcal{R} \subseteq \{1,\ldots,d\}$; note that $|\mathcal{R}|=h$. Then

$$\text{rank}(\phi_j(H)) \geq \sum_{r\in\mathcal{R}} \delta(j,r),$$

so

$$\sum_{j=1}^m s_j \,\text{rank}(\phi_j(H)) \geq \sum_{j=1}^m s_j \sum_{r\in\mathcal{R}} \delta(j,r) = \sum_{r\in\mathcal{R}} \sum_{j=1}^m s_j\delta(j,r) \geq \sum_{r\in\mathcal{R}} \text{rank}(H_r) = \sum_{r\in\mathcal{R}} 1 = |\mathcal{R}| = \text{rank}(H).$$

□

Remark 6.7. *We can generalize this to the case where the ϕ_j do not choose subsets S_j of the indices $i_1,\ldots,i_d$, but rather subsets of suitable independent linear combinations of the indices, for example subsets of $\{i_1, i_1+i_2, i_3-2i_1\}$ instead of subsets of $\{i_1,i_2,i_3\}$. We will discuss recognizing the product case in more general array references in Part 2 of this paper.*

We now give a number of concrete examples. We introduce a common notation from linear programming that we will also use later. We start with the version of the hypothesis (6.2) used in the proof, $1 \leq \sum_{j=1}^m s_j\delta(j,i)$, and rewrite this as $\mathbf{1}_d^T \leq s^T \cdot \Delta$, where $\mathbf{1}_d \in \mathbb{R}^d$ is a column vector of all ones, $s=(s_1,\ldots,s_m)^T \in \mathbb{R}^m$ is a column vector, and $\Delta \in \mathbb{R}^{m\times d}$ is a matrix with $\Delta_{ji}=\delta(j,i)$. Recall that the communication lower bound in Section 4 depends on $\sum_{j=1}^m s_j = \mathbf{1}_m^T \cdot s$, and that the lower bound is tightest (maximal) when $\mathbf{1}_m^T \cdot s$ is minimized. This leads to the linear program

$$\text{minimize } \mathbf{1}_m^T \cdot s \text{ subject to } \mathbf{1}_d^T \leq s^T \cdot \Delta, \tag{6.3}$$

where we refer to the optimal value of $\mathbf{1}_m^T \cdot s$ as s_{HBL}. Note that (6.3) is not necessarily feasible, in which case s_{HBL} does not exist. This is good news, as the matrix powering example shows. Since Δ has one row per ϕ_j (i.e., per array reference), and one column per loop index, we will correspondingly label the rows and columns of the Δ matrices below to make it easy to see how they arise. These lower bounds are all attainable (see Section 7.2).

Example: Matrix-Vector Multiplication (Part 2/3). We continue the example from Section 6.2.2, with $\phi_y(i_1,i_2)=(i_1)$, $\phi_A(i_1,i_2)=(i_1,i_2)$, and $\phi_x(i_1,i_2)=(i_2)$, or

$$\Delta = \begin{array}{c} \\ y \\ A \\ x \end{array} \begin{array}{c} \begin{array}{cc} i_1 & i_2 \end{array} \\ \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{pmatrix} \end{array},$$

and the linear program (6.3) becomes minimizing $s_{\text{HBL}}=s_y+s_A+s_x$ subject to $s_y+s_A\geq 1$ and $s_A+s_x\geq 1$. The solution is $s_x=s_y=0$ and $s_A=1=s_{\text{HBL}}$. This yields a communication lower bound of $\Omega(N^2/M^{s_{\text{HBL}}-1})=\Omega(N^2)$. This reproduces the well-known result that there is no opportunity to minimize communication in matrix-vector multiplication (or other so-called BLAS-2 operations) because each entry of A is used only once. As we will see in Section 7.1.2, this trivial lower bound (no data reuse) is easily attainable. △

Example: N-Body Simulation and Database Join (Part 1/3). The (simplest) code that accumulates the force on each particle (body) due to all N particles is

```
for $i_1 = 1 : N$,    for $i_2 = 1 : N$,
    $F(i_1)$ += compute_force($P(i_1), P(i_2)$)
```

We get $\phi_F(i_1,i_2)=(i_1)$, $\phi_{P_1}(i_1,i_2)=(i_1)$, and $\phi_{P_2}(i_1,i_2)=(i_2)$, or

$$\Delta = \begin{array}{c} \\ F \\ P_1 \\ P_2 \end{array} \begin{array}{c} \begin{array}{cc} i_1 & i_2 \end{array} \\ \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array}.$$

Now the solution of the linear program is not unique: $s_F=\alpha$, $s_{P_1}=1-\alpha$ and $s_{P_2}=1$ is a solution for any $0\leq\alpha\leq 1$, all yielding $s_{\text{HBL}}=2$ and a communication lower bound of $\Omega(N^2/M)$.

Consider also the similar program

```
for $i_1 = 1 : N_1$,    for $i_2 = 1 : N_2$,
    if predicate$(R(i_1), S(i_2))$ = true,
        output$(i_1, i_2)$ = func$(R(i_1), S(i_2))$
```

which represents a generic "nested loop" database join algorithm of the sets of tuples R and S with $|R| = N_1$ and $|S| = N_2$. We have a data-dependent branch in the inner loop, so we split the iteration space $\mathcal{Z} =: \mathcal{Z}_{\text{true}} + \mathcal{Z}_{\text{false}}$ depending on the value of the predicate (we still assume that the predicate is evaluated for every (i_1, i_2)). This gives

$$\Delta_{\text{true}} = \begin{array}{c} \\ R \\ S \\ \text{output} \end{array} \begin{array}{c} \begin{array}{cc} i_1 & i_2 \end{array} \\ \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \end{array} \quad \text{and} \quad \Delta_{\text{false}} = \begin{array}{c} \\ R \\ S \end{array} \begin{array}{c} \begin{array}{cc} i_1 & i_2 \end{array} \\ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array},$$

leading to $s_{\text{HBL,true}} = 1$ and $s_{\text{HBL,false}} = 2$, respectively. Writing $|\mathcal{Z}_{\text{true}}| = \alpha|\mathcal{Z}| = \alpha N_1 N_2$, then we obtain lower bounds $\Omega(\alpha N_1 N_2)$ and $\Omega((1-\alpha) N_1 N_2 / M)$ for iterations $\mathcal{Z}_{\text{true}}$ and $\mathcal{Z}_{\text{false}}$, respectively. So we can take the maximum of these two lower bounds to obtain a lower bound for the whole program. Altogether, this yields

$$\max\left(\Omega(\alpha N_1 N_2), \Omega((1-\alpha) N_1 N_2 / M)\right) = \Omega(\max(\alpha N_1 N_2, N_1 N_2 / M)),$$

an increasing function of $\alpha \in [0, 1]$ (using the fact that $(M + 1)/M = \Theta(1)$). When α is close enough to zero, we have the 'best' lower bound $\Omega(N_1 N_2 / M)$, and when α is close enough to 1, we have $\Omega(\alpha N_1 N_2)$, the size of the output, a lower bound for any computation. $\triangle$

Example: Matrix Multiplication (Part 3/5). We outlined this result already in part 2/5 of this example (see Section 1). We have $\phi_A(i_1, i_2, i_3) = (i_1, i_3)$, $\phi_B(i_1, i_2, i_3) = (i_3, i_2)$, and $\phi_C(i_1, i_2, i_3) = (i_1, i_2)$, or

$$\Delta = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} i_1 & i_2 & i_3 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}.$$

One may confirm that the corresponding linear program has solution $s_A = s_B = s_C = 1/2$ so $s_{\text{HBL}} = 3/2$, yielding the well known communication lower bound of $\Omega(N^3/M^{1/2})$. Recall from Section 2 that the problem of matrix multiplication was previously analyzed in [13] using the Loomis-Whitney inequality (Theorem 2.1, a special case of Theorem 3.2) and extended to many other linear algebra algorithms in [4]. $\triangle$

Example: Tensor Contraction (Part 1/2). Assuming $1 \leq j < k - 1 < d$, the code is

```
for $i_1 = 1 : N$,    ...,    for $i_d = 1 : N$,
    $C(i_1, \ldots, i_j, i_k, \ldots, i_d)$ += $A(i_1, \ldots, i_{k-1}) \cdot B(i_{j+1}, \ldots, i_d)$
```

We get

$$\Delta = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccccccccc} i_1 & \cdots & i_j & i_{j+1} & \cdots & i_{k-1} & i_k & \cdots & i_d \end{array} \\ \begin{pmatrix} 1 & \cdots & 1 & 1 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 1 & \cdots & 1 & 1 & \cdots & 1 \\ 1 & \cdots & 1 & 0 & \cdots & 0 & 1 & \cdots & 1 \end{pmatrix} \end{array}.$$

The linear program is identical to the linear program for matrix multiplication, so $s_{\text{HBL}} = 3/2$, and communication lower bound is $\Omega(N^d/M^{1/2})$. $\triangle$

Example: Complicated Code (Part 2/4). We already outlined this example in part 1/4 (see Section 1). The code given in part 1/4 leads to

$$\Delta = \begin{array}{c} \\ A_1 \\ A_2 \\ A_{3,1} \\ A_4, A_{3,2} \\ A_5 \\ A_6 \end{array} \begin{array}{c} \begin{array}{cccccc} i_1 & i_2 & i_3 & i_4 & i_5 & i_6 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array},$$

since A_3 is accessed twice, once with the same subscript as A_4, so we have ignored the duplicate subscript following Remark 6.2. We obtain $s = (2/7, 1/7, 3/7, 2/7, 3/7, 4/7)^T$ as a solution, giving $s_{\mathrm{HBL}} = 15/7$, so the communication lower bound is $\Omega(N^6/M^{8/7})$. △

Example: Matrix Powering (Part 3/4). We continue this example from part 2/4 (above), with ϕ_A, ϕ_B and ϕ_C as given there. So we have

$$\Delta = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{cccc} i_1 & i_2 & i_3 & i_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

The first column of Δ gives us the infeasible constraint $0 \cdot s_A + 0 \cdot s_B + 0 \cdot s_C \geq 1$. As mentioned above, we can reorganize the loops so that we can increase k arbitrarily without communication. The issue is that the intermediate powers of A do not occupy unique memory addresses, but rather overwrite the previous power (stored in the buffers B and C). Following the strategy of imposing Reads and Writes, we apply Theorem 6.6 to the second code fragment in part 1/4 of this example (see Section 2), yielding

$$\Delta = \begin{array}{c} \\ \hat{A}_1 \\ \hat{A}_2 \\ \hat{A}_3 \end{array} \begin{array}{c} \begin{array}{cccc} i_1 & i_2 & i_3 & i_4 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

where the subscripts on $\hat{A}$ distinguish its three appearances. The resulting linear program is feasible with $s_1 = s_2 = s_3 = 1/2$ and $s_{\mathrm{HBL}} = 3/2$, yielding a communication lower bound of $ckN^3/M^{1/2}$ for some constant $c > 0$. As mentioned in part 1/4, the number of imposed Reads and Writes is at most $2(k-2)N^2$, so subtracting we get a communication lower bound for the original algorithm of $ckN^3/M^{1/2} - 2(k-2)N^2$. When $N = \omega(M^{1/2})$, the first dominates, and we see the complexity is the same as k independent matrix multiplications. When $N = O(M^{1/2})$, the second term may dominate, leading to a zero lower bound, which reflects the fact that one can read the entire matrix A into fast memory, perform the entire algorithm while computing and discarding intermediate powers of A, and only write out the final result A^k. In other words, any communication lower bound that is proportional to the number of loop iterations $|\mathcal{Z}| = (k-1)N^3$ must be zero. △

7 Communication-optimal algorithms

Our ultimate goal is to take a simple expression of an algorithm, say by nested loops, and a target architecture, and automatically generate a communication optimal version, i.e., that attains the lower bound of Theorem 4.1, assuming the dependencies between loop iterations permit. In this section we present some special cases where this is possible, corresponding to the cases in Section 6 where we could effectively compute the linear constraints (3.1).

The rest of this section is organized as follows. Section 7.1 briefly addresses the infeasible and injective cases described in Section 6.2. Section 7.2 presents communication-optimal sequential algorithms for the product case whose lower bound was presented in Section 6.3. Section 7.3 presents communication-optimal parallel algorithms for the same product case, including generalizations of recent algorithms like 2.5D matrix multiplication [20], which use additional memory (larger M) to replicate data and reduce communication. In Part 2 of this paper (outlined in Section 8) we will discuss attainability in general.

7.1 Trivial Cases

7.1.1 Infeasible Case

As shown in Lemma 6.4, if $H = \bigcap_{j=1}^{m} \ker(\phi_j)$ is nontrivial, then s_{HBL} doesn't exist. That is, since H is infinite, we can potentially perform an unbounded number of iterations $\mathcal{I} \in H$ with the same m operands (which fit in cache, since $m = o(M)$ by assumption — see Section 4).

Recall the matrix powering example from the previous section (parts 2/4 and 3/4). In part 2/4, we saw a lower bound of 0 words moved was possible, provided the matrix dimension N was bounded in terms of $M^{1/2}$. We then showed in part 3/4 how to obtain a lower bound on the modified code with imposed Reads and Writes (from part 1/4, in Section 4) which is sometimes tighter, depending on the relative sizes of N and M. In part 4/4 below, we show that the tighter bound is also attainable, by reorganizing the modified code.

7.1.2 Injective Case

As shown in Lemma 6.5, if there is an injective ϕ_j, we have $s_{\mathrm{HBL}} = 1$. Thus no data reuse is possible asymptotically. So, executing the loop nest in any order (of the iteration space) will attain the lower bound. We revisit the matrix-vector multiplication example from the previous section (parts 1/3 and 2/3) in part 3/3 below.

7.2 Product Case — Sequential Algorithms

Here we show how to find an optimal sequential algorithm in the special case discussed in Theorem 6.6, where the subscripts ϕ_j all choose subsets S_j of the loop indices $i_1, \ldots, i_d$. Recalling the notation from Section 6.3, (6.3) provided a linear program for $s = (s_1, \ldots, s_m)^T$ (that we call "sLP" here),

$$\text{(sLP)} \quad \text{minimize } \mathbf{1}_m^T \cdot s \text{ subject to } \mathbf{1}_d^T \leq s^T \cdot \Delta,$$

where $\Delta \in \mathbb{R}^{m \times d}$ was a matrix with $\Delta_{ji} = 1$ if $i \in S_j$ and 0 otherwise. We let s_{HBL} denote the minimum value of $\mathbf{1}_m^T \cdot s$.

For notational simplicity, we consider a computer program of the form

```
for $i_1 = 1 : N$,   for $i_2 = 1 : N$,   ...,   for $i_d = 1 : N$,
    inner_loop($i_1, i_2, \ldots, i_d$)
```

and ask how to reorganize it to attain the communication lower bound of $\Omega(N^d/M^{s_{\mathrm{HBL}}-1})$ in a sequential implementation. As explained in Section 4, explicit loop nests are not necessary, but it is easier to explain the result.

We consider a reorganization of the following "blocked" or "tiled" form:

```
for $j_1 = 1 : M^{x_1} : N$,   for $j_2 = 1 : M^{x_2} : N$,   ...,   for $j_d = 1 : M^{x_d} : N$,
    for $k_1 = 0 : M^{x_1} - 1$,   for $k_2 = 0 : M^{x_2} - 1$,   ...,   for $k_d = 0 : M^{x_d} - 1$,
        $(i_1, i_2, \ldots, i_d) = (j_1, j_2, \ldots, j_d) + (k_1, k_2, \ldots, k_d)$
        inner_loop($i_1, i_2, \ldots, i_d$)
```

Here M is the fast memory size, and $x_1, \ldots, x_d$ are parameters to be determined below.

Here we show that simply solving the dual linear program of (sLP) provides the values of $x_1, \ldots, x_d$ that we need for the tiled code to be optimal, i.e., to attain the communication lower bound:

Theorem 7.1. *Suppose that the linear program (sLP) for s is feasible. Then the dual linear program "xLP" for $x = (x_1, \ldots, x_d)^T$*

$$\textit{(xLP)} \quad \textit{maximize } \mathbf{1}_d^T \cdot x \textit{ subject to } \mathbf{1}_m \geq \Delta \cdot x,$$

is also feasible, and using these values of x in the tiled code lets it attain the communication lower bound of $\Omega(N^d/M^{s_{\mathrm{HBL}}-1})$ words moved.

Proof. By duality the solution x of (xLP) satisfies $\mathbf{1}_d^T \cdot x = \mathbf{1}_m^T \cdot s = s_{\mathrm{HBL}}$. Let $E = \{(i_1, \ldots, i_d)\} \subset \mathbb{Z}^d$ be the set of lattice points traversed by the d innermost loops of the tiled code, i.e., the loops over $k_1, \ldots, k_d$, for one value of $(j_1, \ldots, j_d)$. It is easy to see that the size of E is $|E| = \prod_{i=1}^d M^{x_i} = M^{\sum_{i=1}^d x_i} = M^{s_{\mathrm{HBL}}}$. Furthermore, the constraint $\mathbf{1}_m \geq \Delta \cdot x$ means that for each j, $1 \geq \sum_{i=1}^d \Delta_{ji} x_i = \sum_{i \in S_j} x_i$, with equality for at least one j. Thus $|\phi_j(E)| = \prod_{i \in S_j} M^{x_i} = M^{\sum_{i \in S_j} x_i} \leq M^1 = M$. In other words, the number of words accessed by the j^{th} array in the innermost d loops is at most M, with equality at least once. Thus the inner d loops of the algorithm perform the maximum possible $M^{s_{\mathrm{HBL}}}$ iterations while accessing $\Theta(M)$ words of data, so the algorithm is optimal. □

We note that in practice, we may want to choose the block sizes M^{x_i} to be a constant factor smaller, in order for all the data accessed in the innermost d loops to fit in fast memory memory simultaneously. We will look at these constants in Part 2 of this work.

We look again at the examples from Section 6.3, using the notation introduced there.

Example: Matrix-Vector Multiplication (Part 3/3). We write $x = (x_1, x_2)^T$ as the unknown in the dual linear program (xLP), so that M^{x_1} is the tile size for variable i_1 and M^{x_2} is the tile size for variable i_2. Then (xLP) becomes maximizing $s_{\mathrm{HBL}} = x_1 + x_2$ subject to $x_1 \leq 1$, $x_1 + x_2 \leq 1$, and $x_2 \leq 1$. The solution is $x_1 = \alpha$ and $x_2 = 1 - \alpha$ for any $0 \leq \alpha \leq 1$, and $s_{\mathrm{HBL}} = 1$. In other words, we can "tile" the matrix A into any M^α–by–$M^{1-\alpha}$

rectangular blocks of total size M. Indeed, such flexibility is common when one of the subscripts is injective (see Section 7.1.2, above).

We can use our approach to further reduce communication, potentially up to a factor of 2, by choosing the optimal α. We simply ignore the array A, and apply the theory just to the vectors x and y, leading to the same analysis as the N–body problem, which tells use to use blocks of size M for both x and y (see next example). We still read each entry of A once (moving N^2 words), but reuse each entry of x and y M times. △

Example: N-Body Simulation and Database Join (Part 2/3). We write $x = (x_1, x_2)^T$ as the unknown in the dual linear program (xLP), so that M^{x_1} is the tile size for variable i_1 and M^{x_2} is the tile size for variable i_2. Then (xLP) becomes maximizing $s_{\text{HBL}} = x_1 + x_2$ subject to $x_1 \le 1$ and $x_2 \le 1$. The solution is $x_1 = x_2 = 1$ and $s_{\text{HBL}} = 2$, which tells us to to take M particles indexed as $P(i_1)$, M (usually different) particles indexed as $P(i_2)$, and compute all M^2 pairs of interactions.

For the database join example, we use the optimal blocking for the iterations $\mathcal{Z}_{\text{false}}$, which also yields $x_1 = x_2 = 1$, and so compute predicate$(R(i_1), S(i_2))$ M^2 times using M values each of R_{i_1} and S_{i_2}. Whenever the predicate is true, output(i_1, i_2) is immediately written to slow memory. Thus there are N_1N_2/M reads of $R(i_1)$ and $S(i_2)$, and $|\mathcal{Z}_{\text{true}}| = \alpha N_1 N_2$ writes of output(i_1, i_2), which attains the lower bound. △

Example: Matrix Multiplication (Part 4/5). We write $x = (x_1, x_2, x_3)^T$ as the unknown in the dual linear program (xLP), so that M^{x_1} is the tile size for variable i_1, M^{x_2} is the tile size for variable i_2, and M^{x_3} is the tile size for variable i_3. Then (xLP) becomes maximizing $s_{\text{HBL}} = x_1 + x_2 + x_3$ subject to $x_1 + x_3 \le 1$, $x_1 + x_2 \le 1$ and $x_2 + x_3 \le 1$. The solution is $x_1 = x_2 = x_3 = 1/2$ and $s_{\text{HBL}} = 3/2$, which tells us the well-known result that each matrix A, B, and C should be tiled into square blocks of size $M^{1/2}$–by–$M^{1/2}$. When M is the cache size on a sequential machine, this is a well-known algorithm, with a communication cost of $\Theta(N^3/M^{1/2})$ words moved between cache and main (slower) memory. △

Example: Tensor Contraction (Part 2/2). We write $x = (x_1, \ldots, x_d)^T$ as the unknown in the dual linear program (xLP), so that M^{x_r} is the tile size for variable i_r. Then (xLP) becomes maximizing $s_{\text{HBL}} = \sum_{r=1}^{d} x_r$ subject to

$$\sum_{r=1}^{k-1} x_r \le 1, \quad \sum_{r=j+1}^{d} x_r \le 1, \quad \text{and} \quad \sum_{r=1}^{j} x_r + \sum_{r=k}^{d} x_r \le 1.$$

The solution is any set of values of $0 \le x_r \le 1$ satisfying

$$\sum_{r=1}^{j} x_r = 1/2, \quad \sum_{r=j+1}^{k-1} x_r = 1/2, \quad \text{and} \quad \sum_{r=k}^{d} x_r = 1/2,$$

so $s_{\text{HBL}} = 3/2$. For example, one could use

$$x_1 = \cdots = x_j = \frac{1}{2j}, \quad x_{j+1} = \cdots = x_{k-1} = \frac{1}{2(k-j-1)}, \quad \text{and} \quad x_k = \cdots = x_d = \frac{1}{2(d-k+1)}$$

which is a natural generalization of matrix multiplication. △

Example: Complicated Code (Part 3/4). We write $x = (x_1, \ldots, x_6)^T$ as the unknown in the dual linear program (xLP), so that M^{x_r} is the tile size for variable i_r. The solution is

$$x = (2/7, 3/7, 1/7, 2/7, 3/7, 4/7)^T$$

(and $s_{\text{HBL}} = 15/7$), leading to the block sizes $M^{2/7}$–by–$\cdots$–by–$M^{4/7}$. △

Example: Matrix Powering (Part 4/4). While (sLP) is infeasible, (xLP) has unbounded solutions. As suggested above, the trivial lower bound of 0 words moved may not always be attainable. After imposing Reads and Writes, we saw in part 3/4 (see previous section) that when N is sufficiently large compared to $M^{1/2}$, the communication lower bound (for the modified program) looks like k independent matrix multiplications, and when $N = O(M^{1/2})$, the lower bound degenerates to 0. To attain the lower bound for the modified code with imposed Reads and Writes, we implement the matrix multiplication $\hat{A}(i_1, \cdot, \cdot) = \hat{A}(1, \cdot, \cdot) \cdot \hat{A}(i_1 - 1, \cdot, \cdot)$ in the inner loop in the optimized way as described in part 4/4 of the matrix multiplication example (above), and leave the outer loop (over i_1) unchanged. △

7.3 Product Case — Parallel Algorithms

We again consider the special case discussed in Theorem 6.6, and show how to attain a communication optimal parallel algorithm. We again start with the unblocked code of the last section, and show how to map it onto P processors so that it is both *load balanced*, i.e., every processor executes inner_loop$(i_1, i_2, \ldots, i_d)$ $O(N^d/P)$ times, and every processor attains the communication lower bound $\Omega((N^d/P)/M^{s_{\mathrm{HBL}}-1})$, i.e., communicates at most this many words to or from other processors. We need some assumption about load balance, since otherwise one processor could store all the data and do all the work with no communication. In contrast to the sequential case, M is no longer fixed by the hardware (e.g., to be the cache size), but can vary from the minimum needed to store all the data in the arrays accessed by the algorithm to larger values. Since the lower bound is a decreasing function of M, we should ideally have a set of algorithms that communicate less as M grows (up to some limit). There is in fact a literature on algorithms for linear algebra [20, 19, 23, 18] and the N–body problem [8] that do attain the lower bound as M grows; here we will see that this is possible in general (again, loop dependencies permitting).

In Section 1, we said that we would only count the number of words moved between processors, not other costs like message latency, network congestion, or costs associated with noncontiguous data. We will also ignore dependencies. This will greatly simplify our presentation, but leave significant work to design more practical parallel algorithms; we leave this until Part 2 of this paper.

Assume for a moment that we have chosen M; we distribute the work among the processors as follows: Supposing that the linear program (sLP) is feasible, let $x = (x_1, \ldots, x_d)^T$ be the solution of the dual linear program (xLP) defined in Theorem 7.1. Just as in the sequential case, we take the lattice of N^d points representing inner loop iterations, indexed from $\mathcal{I} = (i_1, i_2, \ldots, i_d) = (0, 0, \ldots, 0)$ to $(N-1, N-1, \ldots, N-1)$, and partition the axis indexed by i_j into N/M^{x_j} contiguous segments of equal size M^{x_j}. (As before, for simplicity we assume all fractions like N/M^{x_j} divide evenly.) This in turn divides the entire lattice into $\prod_{i=1}^{d} N/M^{x_i} = N^d/M^{s_{\mathrm{HBL}}}$ "bricks" of $M^{s_{\mathrm{HBL}}}$ lattice points each. Let E refer to the lattice points in one brick. Just as in the proof of Theorem 7.1, the amount of data from array A_j needed to execute the inner loop iterations indexed by E is bounded by $|\phi_j(E)| \le M$. So, up to a constant multiple m (which we ignore), each processor has enough memory to store the data needed to execute a brick. This yields one constraint from our approach: to be load balanced, there need to be at least P bricks, i.e., $N^d/M^{s_{\mathrm{HBL}}} \ge P$, or $M \le N^{d/s_{\mathrm{HBL}}}/P^{1/s_{\mathrm{HBL}}}$. In other words, we should pick M no larger than this upper bound in order to use all available processors. (Of course M also cannot exceed the memory available on each processor.)

The next constraint arises from having enough memory in all the processors to hold all the data. Recalling the notation used to define (sLP), we let S_j be the subset of loop indices appearing in ϕ_j, and we have $|S_j| = d_j = \mathrm{rank}(\phi_j(\mathbb{Z}^d))$. This means that the total storage required by array A_j is N^{d_j}, and the total storage required by all arrays is $\sum_{j=1}^{m} N^{d_j}$, yielding the lower bound $M \ge \sum_{j=1}^{m} N^{d_j}/P$, and so altogether

$$\frac{N^{d/s_{\mathrm{HBL}}}}{P^{1/s_{\mathrm{HBL}}-1}} \ge M \ge \frac{\sum_{j=1}^{m} N^{d_j}}{P} \ge \frac{N^{d_{\max}}}{P}, \tag{7.1}$$

where $d_{\max} := \max_{j=1}^{m} d_j$. Plugging these bounds on M into the communication lower bound yields

$$\frac{N^{d/s_{\mathrm{HBL}}}}{P^{1/s_{\mathrm{HBL}}}} \le \frac{N^d/P}{M^{s_{\mathrm{HBL}}-1}} \le \frac{N^{d-d_{\max}(s_{\mathrm{HBL}}-1)}}{P^{2-s_{\mathrm{HBL}}}}. \tag{7.2}$$

The expression on the left is simply the memory-independent lower bound $\Omega((|\mathcal{Z}|/P)^{1/s_{\mathrm{HBL}}})$ discussed in Section 4.6. Note that when M equals its upper bound, it also equals the communication lower bound, which means that the content of the memory only needs to be communicated once; this is related to the fact that the assumption $|\mathcal{Z}|/P = \omega(F)$ fails to hold (see Section 4.6).

For example, for matrix multiplication ($d = 3$, $s_{\mathrm{HBL}} = 3/2$, and $d_{\max} = 2$) we get the familiar constraints $N^2/P^{2/3} \ge M \ge N^2/P$ and $N^2/P^{2/3} \le (N^3/P)/M^{1/2} \le N^2/P^{1/2}$ [3, 1, 13, 20].

For M in this range, we may express our parallel algorithm most simply as follows:

```
while there are unexecuted bricks
    assign $P$ unexecuted bricks to $P$ processors
    in parallel, communicate the needed data to each processor
    in parallel, execute each brick
```

The algorithm is clearly load balanced (and oblivious to any loop dependencies). It remains to compute the communication complexity: this is simply the number of times each processor executes a brick, times the communication required per brick. Since there are $N^d/M^{s_{\text{HBL}}}$ bricks shared evenly by P processors, with $O(M)$ communication per brick, the result is $O((N^d/P)/M^{s_{\text{HBL}}-1})$, i.e., the lower bound is attained. More formally, we could use an abstract model of computation like LPRAM [1] or BSPRAM [22, 18] to describe this algorithm.

Instead, by making more assumptions (satisfied by some of our running examples) we may also describe the algorithms in more detail as to how to map them to an actual machine (we will weaken these assumptions in Part 2). We will assume that $\Delta\cdot x = \mathbf{1}_m$, not just $\Delta\cdot x \le \mathbf{1}_m$ as required in (xLP). By choosing all $x_i = 1/d_{\max}$, which satisfies $\Delta\cdot x \le \mathbf{1}_m$, we see that $s_{\text{HBL}} \ge \mathbf{1}_d^T\cdot x = d/d_{\max}$. We will in fact assume $s_{\text{HBL}} = d/d_{\max}$. Finally, we will assume for convenience that certain intermediate expressions, like $P^{s_{\text{HBL}}}$, are integers. We will write $M = CN^{d_{\max}}/P$, where C is the number of copies of the data we will use; $C = 1$ corresponds to the minimum memory necessary. Combining with the upper bound on M from (7.1), we get $N^{d/s_{\text{HBL}}}/P^{1/s_{\text{HBL}}} \ge CN^{d_{\max}}/P$, or $C \le P^{1-1/s_{\text{HBL}}}$.

Next, we divide the P processors into C groups of P/C processors each. The total number of bricks is $\prod_{i=1}^d N/M^{x_i} = N^d/M^{s_{\text{HBL}}} = N^d/(CN^{d_{\max}}/P)^{s_{\text{HBL}}} = (P/C)^{s_{\text{HBL}}}$. If we used just P/C processors to process all these bricks (P/C at a time in parallel), then it would take $(P/C)^{s_{\text{HBL}}-1}$ parallel phases. But since there are C groups of processors, only $P^{s_{\text{HBL}}-1}/C^{s_{\text{HBL}}}$ phases are needed.

Now we describe how the data is initially laid out in each group of P/C processors. Since we assume there is enough memory $P\cdot M$ for C copies of the data, each group of P/C processors will have its own copy. For each array A_μ, $1 \le \mu \le m$, we index the processors by $(j_k \in \mathbb{N} : 0 \le j_k < N/M^{x_k})_{k\in S_\mu}$, so the number of processors indexed is $\prod_{k\in S_\mu} N/M^{x_k} = N^{d_\mu}/M^{\sum_{k\in S_\mu} x_k} = N^{d_\mu}/M$, since we have assumed $\Delta\cdot x = \mathbf{1}_m$. Finally, $N^{d_\mu}/M \le N^{d_{\max}}/M = P/C$, so we index all the processors in the group when $d_\mu = d_{\max}$. (When $d_\mu < d_{\max}$, i.e., the array A_μ has fewer than the maximum number of dimensions, then it is smaller and so needs fewer processors to store.) Since A_μ has subscripts given by the indices in S_μ, we may partition A_μ by blocking the subscripts i_k, for $k \in S_\mu$, into blocks of size M^{x_k}, so that each block contains $\prod_{k\in S_\mu} M^{x_k} = M$ entries, and so may be stored on a single processor. Then block $(j_k)_{k\in S_\mu}$ is stored on the processor with the same indices, for $0 \le j_k < N/M^{x_k}$. This indexing will make it easy for a processor to find the data it needs, given the brick it needs to execute.

Next we describe how to group bricks (of the iteration space), indexed by $\mathcal{J} = (j_k \in \mathbb{N} : 0 \le j_k < N/M^{x_k})_{k=1}^d$, into groups of P/C to be done in each phase. We define a "superbrick" to consist of $(N/M^{x_k})^{1/s_{\text{HBL}}}$ contiguous bricks in each direction, containing a total of $\prod_{k=1}^d (N/M^{x_k})^{1/s_{\text{HBL}}} = P/C$ bricks. Thus one superbrick can be executed in one parallel phase by one group of P/C processors, and C superbricks can be executed in one parallel phase by all P. Given the indexing described in the last paragraph, the index $\mathcal{J}$ of each brick in a superbrick indicates exactly which processor in a group owns the data necessary to execute it.

We still need to assign superbricks to groups of processors. There are many ways to do this. We consider two extreme cases, $C = 1$ and $C = P^{1-1/s_{\text{HBL}}}$.

When $C = 1$, there is just one group of processors, and there are $P^{s_{\text{HBL}}-1}$ superbricks, each containing P bricks, with each brick assigned to a processor using the index $\mathcal{J}$. Altogether there are $P^{s_{\text{HBL}}-1}$ parallel phases to complete the execution. We emphasize that this is just one way of many to accomplish parallelization; see the examples below. For example, for matrix multiplication, this corresponds to the classical so-called '2D' algorithms, where the P processors are laid out in a $P^{1/2}$–by–$P^{1/2}$ grid.

When C equals its maximum value $C = P^{1-1/s_{\text{HBL}}}$, then there are just C superbricks and the number of parallel phases is $P^{s_{\text{HBL}}-1}/C^{s_{\text{HBL}}} = 1$. For example, for matrix multiplication this corresponds to a so-called '3D' algorithm, where the P processors are laid out in a $P^{1/3}$–by–$P^{1/3}$–by–$P^{1/3}$ grid (see [1]). For the N–body problem, this corresponds to a '2D' algorithm (see [8]).

Only two of our running examples satisfy the assumptions in this special case. As we mentioned above, we will weaken these assumptions in Part 2 to address the general 'product case,' as described in Sections 6.3 and 7.2.

Example: *N*-Body Simulation and Database Join (Part 3/3). Our assumptions in the preceding paragraphs apply to the N–body example, with $x_1 = x_2 = 1$ and $s_{\text{HBL}} = 2 = d/d_{\max}$. (We will not discuss the database join example here because it does not satisfy the assumptions above.) $\triangle$

Example: Matrix Multiplication (Part 5/5). Here our assumptions in the preceding paragraphs apply, with $x_1 = x_2 = x_3 = 1/2$ and $s_{\text{HBL}} = 3/2 = d/d_{\max}$. $\triangle$

Example: Complicated Code (Part 4/4). This example does not satisfy the assumptions of the preceding paragraphs, so we will instead consider the similar-looking code sample

```
for $i_1 = 1 : N$,  for $i_2 = 1 : N$,  for $i_3 = 1 : N$,  for $i_4 = 1 : N$,  for $i_5 = 1 : N$,  for $i_6 = 1 : N$,
    $A_1(i_1, i_3, i_6) \mathrel{+}= \text{func}_1(A_2(i_1, i_2, i_4), A_3(i_2, i_3, i_5), A_4(i_3, i_4, i_6))$
    $A_5(i_2, i_5, i_6) \mathrel{+}= \text{func}_2(A_6(i_1, i_4, i_5), A_3(i_3, i_4, i_6))$
```

where we have added the additional subscript i_5 to the access of A_5, leading to

$$\Delta = \begin{array}{l} \\ A_1 \\ A_2 \\ A_{3,1} \\ A_4, A_{3,2} \\ A_5 \\ A_6 \end{array} \begin{array}{c} \begin{array}{cccccc} i_1 & i_2 & i_3 & i_4 & i_5 & i_6 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

and the solution $x_1 = \cdots = x_6 = 1/3$, so now our previous assumptions are satisfied: $\Delta \cdot x = 1$, and $s_{\text{HBL}} = 2 = d/d_{\max} = 6/3$. $\triangle$

8 Conclusions

Our goal has been to extend the theory of communication lower bounds, and to design algorithms that attain them. Motivated by a geometric approach first used for the 3 nested loops common in linear algebra, we extended these bounds to a very general class of algorithms, assuming only that that the data accessed can be modeled as arrays with subscripts that are linear functions of loop indices. This lets us use a recent result in functional analysis to bound the maximum amount of data reuse (F), which leads to a communication lower bound. The bound is expressed in terms of a linear program that can be written down and solved decidably. For many programs of practical interest, it it possible to design a communication-optimal algorithm that attains the bound. This is true, for example, when all the array subscripts are just subsets of the loop indices. This includes, among other examples, linear algebra, direct N–body interactions, database joins, and tensor contractions.

Part 2 of this paper will both expand the classes of algorithms discussed in Section 6 for which the lower bound can be easily computed, and expand the discussion of Section 7 on the attainability of our lower bounds.

Part 2 will also improve the algorithm in Section 5.1 to improve its efficiency. First, we will show that it suffices to check condition (3.1) for a *subset* $\mathcal{L}$ of subgroups H of $\mathbb{Z}^d$, rather than *all* subgroups. In this discrete extension of [25, Theorem 1.8], previously mentioned in Remark 5.2, $\mathcal{L}$ is the lattice generated by the kernels of the homomorphisms, $\ker(\phi_1), \ldots, \ker(\phi_j)$. We will discuss a few practical cases where $\mathcal{L}$ is finite and the approach simplifies further: this includes the case when there are at most $m = 3$ arrays in the inner loop, as well as a generalization of the product case from Section 6.3.

Then, we discuss a few other cases where $\mathcal{L}$ may be infinite, but we know in advance that we need only check a certain finite subset of subgroups H from $\mathcal{L}$. This includes the case when all the arrays are 1– or $(d-1)$–dimensional, and when there are at most $d = 4$ loops in the loop nest.

We then return to the question of attainability, i.e., finding an optimal algorithm. We discuss two inequalities, (3.2) and $|E| \le M^{s_{\text{HBL}}}$, in which equality must be (asymptotically) attainable to find an optimal algorithm. For the former inequality, we show that asymptotic equality is always attainable by a family of cubes in a critical subgroup. For the latter inequality, we give a special case where asymptotic equality also holds, while leaving the general case open. These results suggest how to reorganize code to obtain an optimal algorithm; we discuss practical concerns, including constants hidden in the asymptotic bounds. We conclude by discussing how our lower bounds depend on the presence of (inter-iteration) data dependencies.

Acknowledgements

We would like to thank Bernd Sturmfels, Grey Ballard, Benjamin Lipshitz, and Oded Schwartz for helpful discussions.

We acknowledge funding from Microsoft (award 024263) and Intel (award 024894), and matching funding by UC Discovery (award DIG07-10227), with additional support from ParLab affiliates National Instruments, Nokia, NVIDIA, Oracle, and Samsung, and support from MathWorks. We also acknowledge the support of the US DOE (grants DE-SC0003959, DE-SC0004938, DE-SC0005136, DE-SC0008700, DE-AC02-05CH11231, DE-FC02-06ER25753, and DE-FC02-07ER25799), DARPA (award HR0011-12-2-0016), and NSF (grants DMS-0901569 and DMS-1001550).

References

[1] A. Aggarwal, A.K. Chandra, and M. Snir. Communication complexity of PRAMs. *Theoretical Computer Science*, 71:3–28, March 1990.

[2] G. Ballard, J. Demmel, and A. Gearhart. Communication bounds for heterogeneous architectures. In *Proceedings of the* 23^{rd} *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2011)*, 2011.

[3] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz. Brief announcement: strong scaling of matrix multiplication algorithms and memory-independent communication lower bounds. In *Proceedings of the* 24^{th} *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2012)*, pages 77–79. ACM, 2012.

[4] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing communication in numerical linear algebra. *SIAM Journal on Matrix Analysis and Applications*, 32(3):866–901, 2011.

[5] C. Bennett and R.C. Sharpley. *Interpolation of Operators*, volume 129 of *Pure and Applied Mathematics*. Academic Press, 1988.

[6] J. Bennett, A. Carbery, M. Christ, and T. Tao. Finite bounds for Hölder-Brascamp-Lieb multilinear inequalities. *Mathematical Research Letters*, 17(4):647–666, 2010.

[7] G.E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages: 2nd GI Conference*, pages 134–183. Springer, 1975.

[8] M. Driscoll, E. Georganas, P. Koanantakool, E. Solomonik, and K. Yelick. A communication-optimal N-body algorithm for direct interactions. Submitted to IPDPS, 2012.

[9] S.H. Fuller and L.I. Millett, editors. *The Future of Computing Performance: Game Over or Next Level?* National Academies Press, 2011.

[10] S.L. Graham, M. Snir, and C.A. Patterson, editors. *Getting up to Speed: the Future of Supercomputing*. National Academies Press, 2005.

[11] P.R. Halmos. *Measure Theory*, volume 18 of *Graduate Texts in Mathematics*. Springer, 2^{nd} printing edition, 1974.

[12] J.-W. Hong and H.T. Kung. I/O complexity: the red-blue pebble game. In *Proceedings of the* 13^{th} *ACM Symposium on Theory of Computing*, pages 326–333. ACM, 1981.

[13] D. Irony, S. Toledo, and A. Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *Journal of Parallel and Distributed Computing*, 64(9):1017–1026, 2004.

[14] J. Koenigsmann. Defining $\mathbb{Z}$ in $\mathbb{Q}$. arXiv preprint 1011.3424, 2010.

[15] S. Lang. *Algebra*, volume 211 of *Graduate Texts in Mathematics*. Springer, 3^{rd} edition, 2002.

[16] L.H. Loomis and H. Whitney. An inequality related to the isoperimetric inequality. *Bulletin of the American Mathematical Society*, 55:961–962, 1949.

[17] Y. Matiyasevich. *Hilbert's Tenth Problem*. MIT Press, 1993.

[18] W.F. McColl and A. Tiskin. Memory-efficient matrix multiplication in the BSP model. *Algorithmica*, 24:287–297, 1999.

[19] E. Solomonik, A. Bhatele, and J. Demmel. Improving communication performance in dense linear algebra via topology aware collectives. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2011)*, pages 77:1–77:11, New York, NY, USA, 2011. ACM.

[20] E. Solomonik and J. Demmel. Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms. In *Euro-Par 2011 Parallel Processing*, volume 6853 of *Lecture Notes in Computer Science*, pages 90–109. Springer, 2011.

[21] A. Tarski. A decision method for elementary algebra and geometry. Technical Report R-109, RAND Corporation, 1951.

[22] A. Tiskin. The bulk-synchronous parallel random access machine. *Theoretical Computer Science*, 136:109–130, 1998.

[23] A. Tiskin. Communication-efficient parallel generic pairwise elimination. *Future Generation Computer Systems*, 23(2):179 – 188, 2007.

[24] R. Vakil. Murphy's law in algebraic geometry: badly-behaved deformation spaces. *Inventiones Mathematicae*, 164(3):569–590, 2006.

[25] S.I. Valdimarsson. The Brascamp-Lieb polyhedron. *Canadian Journal of Mathematics*, 62(4):870–888, 2010.