Quantum algorithms for jet clustering

Annie Y. Wei,[1,*] Preksha Naik,[1,†] Aram W. Harrow,[1,‡] and Jesse Thaler[1,2,§]

[1]*Center for Theoretical Physics, Massachusetts Institute of Technology,*
Cambridge, Massachusetts 02139, USA
[2]*Department of Physics, Harvard University, Cambridge, Massachusetts 02138, USA*

Identifying jets formed in high-energy particle collisions requires solving optimization problems over potentially large numbers of final-state particles. In this work, we consider the possibility of using quantum computers to speed up jet clustering algorithms. Focusing on the case of electron-positron collisions, we consider a well-known event shape called thrust whose optimum corresponds to the most jetlike separating plane among a set of particles, thereby defining two hemisphere jets. We show how to formulate thrust both as a quantum annealing problem and as a Grover search problem. A key component of our analysis is the consideration of realistic models for interfacing classical data with a quantum algorithm. With a sequential computing model, we show how to speed up the well-known $O(N^3)$ classical algorithm to an $O(N^2)$ quantum algorithm, including the $O(N)$ overhead of loading classical data from N final-state particles. Along the way, we also identify a way to speed up the classical algorithm to $O(N^2 \log N)$ using a sorting strategy inspired by the SISCone jet algorithm, which has no natural quantum counterpart. With a parallel computing model, we achieve $O(N \log N)$ scaling in both the classical and quantum cases. Finally, we consider the generalization of these quantum methods to other jet algorithms more closely related to those used for proton-proton collisions at the Large Hadron Collider.

I. INTRODUCTION

Jets are collections of collimated, energetic hadrons formed in high-energy particle collisions. With an appropriate choice of jet clustering algorithm [1], jets are a robust probe of quantum chromodynamics and a useful proxy for determining the kinematics of the underlying hard scattering process. The problem of identifying jets from collision data is a nontrivial task, however, since the jet clustering algorithm must be matched to the physics question of interest. Moreover, it is a computationally intensive task, as it often involves performing optimizations over potentially large numbers of final-state particles.

In this paper, we consider the possibility of using quantum computers to speed up jet identification. We focus on the well-known problem of partitioning an electron-positron collision event into two *hemisphere jets*, though our results are relevant for other optimization problems beyond

[*]anniewei@mit.edu
[†]prekshan@mit.edu
[‡]aram@mit.edu
[§]jthaler@mit.edu

high-energy physics. Our main results are summarized in Table I, where the computational scaling is given for N particles in the final state. We show how to improve the well-known $O(N^3)$ classical algorithm [2] to an $O(N^2)$ quantum algorithm, which includes the cost of loading the classical data into a sequential quantum computing architecture. On the other hand, we also show how to speed up the classical algorithm to $O(N^2 \log N)$, using a clever sorting strategy from Ref. [3], which matches the quantum performance up to $\log N$ factors. Finally, using parallel computing architectures, we achieve $O(N \log N)$ scaling in both the classical and quantum cases, albeit for very different computational reasons.

Quantum algorithms have been shown to achieve speedups over classical algorithms [4], resulting, in theory, in time savings which are even more pronounced over large datasets. That said, many proposed quantum algorithms for machine learning tasks often omit considerations that would be needed to actually implement them in practice, such as a strategy to interface classical data with a quantum computing architecture. One solution is to assume the availability of qRAM [5], which would let our quantum computer access a classical dataset in superposition; however, this additional hardware requirement may not be easy to implement in practice. Here, we consider realistic applications of both quantum annealing [6–8] and Grover search [9–11] to jet finding, including the

TABLE I. Summary of classical and quantum thrust algorithms, where the asymptotic scaling is for a single collision event with N particles. All strategies have a classical space overhead of $O(N)$ bits for read access to the classical data. The classical sorting strategies also require write access to $O(N \log N)$ bits. For ease of exposition throughout, we treat each real number as being specified to a constant $O(1)$ bits of precision.

Implementation	Time usage	Qubit usage	Sections
Classical [2]	$O(N^3)$	...	Sec. III A
Classical with sort (using [3])	$O(N^2 \log N)$	...	Sec. III C
Classical with parallel sort	$O(N \log N)$	...	Sec. III D
Quantum annealing	Gap Dependent	$O(N)$	Sec. IV
Quantum search: sequential model	$O(N^2)$	$O(\log N)$	Sec. V C
Quantum search: parallel model	$O(N \log N)$	$O(N \log N)$	Sec. V E

$O(N)$ overhead of loading classical collision data into the quantum computer.

The specific jet finding algorithm we use is based on thrust [12–14]. Thrust is an event shape widely measured in electron-positron collisions [15–27]. The optimum value of thrust defines the most jetlike separating plane among a set of final-state particles, thereby partitioning the event into two hemisphere jets. Algorithmically, it poses an interesting problem because it can be viewed in various equivalent ways—such as a partitioning problem or as an axis-finding problem—which in turn lead to different algorithmic strategies.

We note that practical thrust computations typically involve only 10–1000 particles per event, so the current $O(N^3)$ classical algorithm [2] is certainly adequate to the task. That said, more efficient jet algorithms are of general interest, for example, in the context of active area calculations [28], which can involve up to millions of ghost particles. We also note that the current default jet algorithm at the Large Hadron Collider (LHC) is anti-k_t [29], which already runs in $O(N \log N)$ time [30,31], and it is unlikely that any quantum algorithm can yield a sublinear improvement. On the other hand, anti-k_t is a hierarchical clustering algorithm (i.e., a heuristic), whereas thrust is a global optimization problem, and there are phenomenological contexts where global jet optimization could potentially yield superior physics performance [32,33]; see also Refs. [34–47]. Jet finding via global optimization has not seen widespread adoption, in part because of the computational overhead, and we hope the quantum and improved classical algorithms developed here spur more research on alternative jet finding strategies.

Beyond the specific applications to jet finding, we believe that the broader question of identifying realistic quantum algorithms for optimization problems should be of interest to both the particle physics and quantum computing communities. Indeed, we regard thrust as a warm-up problem for the more general development of quantum algorithms for collider data analysis. (For other quantum algorithms for collider physics, see Refs. [48,49] for Higgs boson identification, Refs. [50,51] for parton shower generation, and Refs. [52–54] for track reconstruction.)

Because collider data are classical (and will likely remain so for the foreseeable future), understanding the limitations imposed by data loading is essential to evaluate the potential of quantum algorithms to speed up or improve data analysis pipelines. At the same time, it is important to assess potential classical improvements to existing collider algorithms, and the sorting strategy of Ref. [3] is an important example of how new classical strategies can sometimes match the gains from quantum computation.

Turning now to an extended outline of this paper, our quantum algorithms build on existing classical strategies to compute thrust. In Sec. II, we define thrust in its various equivalent manifestations, as both a partitioning problem and an axis-finding problem. Then, in Sec. III, we review classical algorithms for computing thrust based on a search over reference axes. As already mentioned, the best known result in the literature requires $O(N^3)$ time [2]. We show how to improve it to $O(N^2 \log N)$ using a sorting strategy inspired by SISCone [3], which appears to have no quantum analog (see Sec. V D).

The first quantum method we consider in Sec. IV involves formulating thrust as a quadratic unconstrained binary optimization (QUBO) problem, which can then be solved via quantum annealing [6,7]. This comes from viewing thrust as a partitioning problem and then considering the brute force enumeration of all candidate partitions. See Refs. [55,56] for other studies of quantum annealing for clustering with unique assignment.

The core results of this paper are in Sec. V, where we describe quantum algorithms for computing thrust based on Grover search [9]. Although naively Grover search offers a square root speedup over any classical search algorithm, in practice Grover search cannot yield sublinear algorithms. The reason is that data loading over a classical database of size N requires $O(N)$ time, which limits the achievable gains. That said, if the classical *search space* scales like $O(N^\alpha)$, we can still use the Grover strategy to reduce the *search loop* to $O(N^{\alpha/2})$, though there will be an additional additive (multiplicative) factor of $O(N)$ if data loading has to happen outside (inside) of the loop. Using the formulation of thrust as a search over reference axes, we show that $\alpha = 2$ in the thrust case. Thus, we can attribute our

speedup to the fact that data loading is performed in superposition, which means that it still requires only $O(N)$ time despite working over a search space of size $O(N^2)$.

The precise speedup achievable in our Grover search strategy depends on the assumed quantum computing paradigm. We implement two models for retrieving and processing the classical data, based on the abstract operations LOOKUP and SUM. The *sequential computing model* requires $\tilde{O}(1)$ qubits and results in an $O(N^2)$ thrust algorithm. Here we use $\tilde{O}(\cdot)$ to mean that we neglect factors that are polylog in N. The *parallel computing model* requires $\tilde{O}(N)$ qubits and results in an $O(N \log N)$ thrust algorithm. Both computing models are applicable to any general problem where the size of the search space scales like $O(N^\alpha)$ with $\alpha \geq 2$, which are precisely the problems that can typically be sped up with a realistic application of Grover search.

In Sec. VI, we assess whether or not there is any quantum advantage for hemisphere jet finding. Formally, if one has read access to $O(N)$ classical bits but only write access to $O(\log N)$ bits, then one cannot implement the classical sorting strategy in Sec, III C. In that case, there is a quantum advantage for both sequential and parallel computing models. With write access to $O(N \log N)$ classical bits, though, classical sorting is possible, and the asymptotic performance of our classical and quantum algorithms is identical (up to $\log N$ factors) in both the sequential and parallel cases. This equivalence appears to be special to algorithms like thrust where the search space scales like $O(N^2)$, and we speculate that larger search spaces might benefit from Grover speedups even if classical sorting is possible.

Finally, in Sec. VII, we briefly consider generalizations of our results to jet algorithms more closely related to those used at the LHC. We consider jet function maximization [43–45], showing that, with suitable modifications, it can be written in QUBO form for quantum annealing. We consider stable cone finding in the spirit of SISCone [3], showing how a single-jet variant we dub SINGLECONE is amenable to quantum search. We also comment on quantum multijet finding motivated by the XCone algorithm

[32,33]. We conclude in Sec. VIII with some broader lessons about quantum algorithms for collider physics.

II. DEFINITION OF THRUST

We start by defining thrust [12–14], noting that it has multiple equivalent definitions that suggest different algorithmic strategies, as shown in Fig. 1. Thrust can be viewed as a partitioning problem, which lends itself naturally to quantum annealing. Thrust can alternatively be viewed as an axis-finding problem, which we can frame as a quantum search problem. Both definitions of thrust can be stated in terms of operator norms, and through this lens, they are in fact dual to each other.

A. Thrust as a partitioning problem

Consider a set of N three-momenta $\{\vec{p}_i\}$ in their center-of-momentum frame, where $\vec{p}_i = \{p_i^x, p_i^y, p_i^z\}$,

$$\sum_{i=1}^{N} \vec{p}_i = 0. \tag{1}$$

An intuitive formulation of thrust (though not exactly the original one [12,13]) is to separate the particles into a partition $H_L \cup H_R$ such that momenta on each side are as "pencil-like" as possible. That is, we seek to maximize the quantity

$$T(H_L) = \frac{2|\sum_{i \in H_L} \vec{p}_i|}{\sum_{i=1}^{N} |\vec{p}_i|} = \frac{2|\sum_{i \in H_R} \vec{p}_i|}{\sum_{i=1}^{N} |\vec{p}_i|}, \tag{2}$$

where the second equality follows from momentum conservation. The quantity known as "thrust" corresponds to the maximum obtainable value,

$$T = \max_{H_L} T(H_L). \tag{3}$$

The factor of 2 in Eq. (2) is conventional such that $1/2 \leq T \leq 1$, where $T = 1$ corresponds to a perfectly pencillike back-to-back configuration and $T = 1/2$ is an isotropic event.
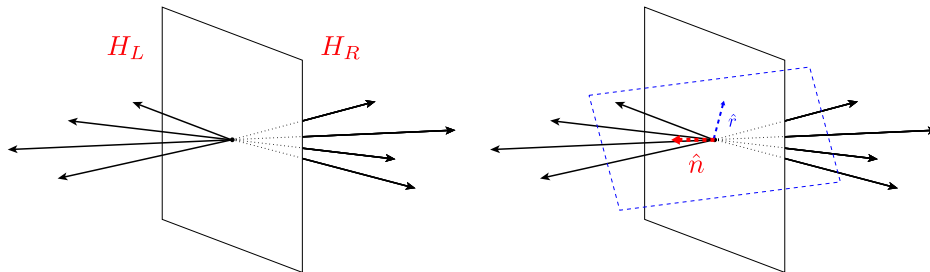

FIG. 1. Two equivalent definitions of thrust as (left) a partitioning problem and (right) an axis-finding problem. The best known classical algorithm is based on plane partitioning via a reference axis $\hat{r}$ (which in general differs from the thrust axis).

There is an equivalent geometric formulation of Eq. (2) due to Ref. [57]. Consider sequentially summing the three-momenta $\{\vec{p}_i\}$ to form a closed polygon. Each sequence yields a different polygon, and computing thrust is equivalent to maximizing twice the diagonal of the polygon over all possible polygons, normalized by the circumference of the polygon. The diagonal splits the polygon into two halves, which yield the partition $H_L \cup H_R$. The particles in H_L are said to be in the "left hemisphere jet" and the particles in H_R are said to be in the "right hemisphere jet."

This definition immediately suggests a naive, brute-force classical strategy for computing thrust. We can enumerate all $O(2^N)$ possible partitions (which can be reduced to $O(2^{N-1})$ using momentum conservation), and then we sum the momenta in each to determine the maximum, resulting in an $O(N2^N)$ algorithm. This is the version of thrust we will use for the quantum annealing formulation in Sec. IV, which corresponds to attacking the problem using quantum brute force.

B. Thrust as an axis-finding problem

An alternative definition of thrust is as an axis-finding problem, which is a bit closer to the historical definition [12,13]. Let $\hat{n}$ be a unit norm vector and define

$$T(\hat{n}) = \frac{\sum_{i=1}^{N} |\hat{n} \cdot \vec{p}_i|}{\sum_{i=1}^{N} |\vec{p}_i|}. \tag{4}$$

Thrust can then be determined by the maximum value of $T(\hat{n})$ over $\hat{n}$,

$$T = \max_{|\hat{n}|=1} T(\hat{n}). \tag{5}$$

The optimal $\hat{n}$ is known as the *thrust axis*,

$$\hat{n}_{\text{opt}} \equiv \underset{|\hat{n}|=1}{\text{argmax}} T(\hat{n}). \tag{6}$$

To gain some intuition for why Eqs. (3) and (5) are equivalent, note that once we find the thrust axis $\hat{n}_{\text{opt}}$, we can partition the particles into those with $\hat{n}_{\text{opt}} \cdot \vec{p}_i > 0$ and those with $\hat{n}_{\text{opt}} \cdot \vec{p}_i < 0$. (It is an interesting bit of computational geometry to show that $\hat{n}_{\text{opt}} \cdot \vec{p}_i$ can never be exactly zero for a finite number of particles.) Said another way, the plane normal to $\hat{n}_{\text{opt}}$ partitions the event into left and right hemispheres. Starting from a nonhemisphere partition, it is always possible to increase the value of thrust in Eq. (2) by flipping a particle from one side to the other, so the optimal partition will be defined by a plane. Because of this equivalence between axis finding and plane partitioning, the thrust objective is sometimes written as

$$T(\hat{n}) = \frac{2 \sum_{i=1}^{N} \Theta(\hat{n} \cdot \vec{p}_i)(\hat{n} \cdot \vec{p}_i)}{\sum_{i=1}^{N} |\vec{p}_i|}, \tag{7}$$

where the Heaviside theta function picks out particles in just one hemisphere.

Note that the optimal partitioning plane is not unique, since there can be multiple planes that yield the same partition. We can exploit this fact to find a computationally convenient partitioning plane, defined by a normal *reference axis* $\hat{r}$. This reference axis will in general be different from the thrust axis $\hat{n}_{\text{opt}}$ but nevertheless yield the same value of thrust via Eq. (2). Specifically, once the optimal partition is known via a reference axis, the thrust axis can be determined from the total three-momentum in the left hemisphere,

$$\hat{n}_{\text{opt}} = \frac{\sum_{i \in H_L} \vec{p}_i}{|\sum_{i \in H_L} \vec{p}_i|}. \tag{8}$$

We will use this reference axis approach for the classical thrust algorithms in Sec. III and for the quantum search strategies in Sec. V.

C. Duality of thrust definitions

Using the formalism of operator norms, we can show that these two definitions of thrust are in fact dual to each other.

Let $M: \mathbf{V} \to \mathbf{W}$ be a map from $\mathbf{V} = \mathbf{R}^m$ with norm $\|\cdot\|_\alpha$ to $\mathbf{W} = \mathbf{R}^n$ with norm $\|\cdot\|_\beta$. The operator norm of M, known as the induced α-to-β norm, is defined as

$$\|M\|_{\alpha \to \beta} \equiv \max_{\|v\|_\alpha = 1} \|Mv\|_\beta. \tag{9}$$

That is, we search over all vectors v in $\mathbf{V}$ with norm 1 and find the maximum norm for the vector Mv in $\mathbf{W}$. The case when α and β are both the usual L^2 norm corresponds to the largest singular value of M, but in general $\|M\|_{\alpha \to \beta}$ can be NP hard to estimate [58]. (Here NP is the class of problems whose solution can be verified in polynomial time, so an NP hard problem is one that is at least as difficult as the hardest problem in NP.) By duality, we can rewrite this as

$$\max_{\|v\|_\alpha = 1} \|Mv\|_\beta = \max_{\|y\|_{\beta^*} = 1} \|M^T y\|_\alpha = \|M^T\|_{\beta^* \to \alpha}, \tag{10}$$

where y is in $\mathbf{W}^*$, the vector space dual to $\mathbf{W}$, defined as $\mathbf{W}^* = \mathbf{R}^n$ with dual norm $\|\cdot\|_{\beta^*}$. Thus, the α-to-β norm of M is the same as the β^*-to-α norm of M^T.

In the context of thrust, we are interested in the following norms for a vector $v \in \mathbf{R}^n$:

$$\|v\|_1 = \sum_i |v_i|, \tag{11}$$

$$\|v\|_2 = \sqrt{\sum_i v_i^2}, \tag{12}$$

$$\|v\|_\infty = \max_i |v_i|. \tag{13}$$

These are known, respectively, as the one-norm, two-norm, and sup-norm. By Hölder's inequality, the space of vectors

endowed with the p norm is dual to the space of vectors endowed with the q norm, where $\frac{1}{p} + \frac{1}{q} = 1$. In particular, the one-norm is dual to the sup-norm, and the two-norm is dual to itself.

Now consider the matrix $M_{ij} = (\vec{p}_i)_j$, whose rows are the N three-momenta and whose columns are the p^x, p^y, and p^z components. This is a map from $\mathbf{R}^3$ to $\mathbf{R}^N$. Letting $\alpha = 2$ and $\beta = 1$, the induced 2-to-1 norm of M is

$$\|M\|_{2\to1} = \max_{\|\hat{n}\|_2=1} \|M\hat{n}\|_1 = \max_{\hat{n}^2=1} \sum_{i=1}^{N} |\hat{n} \cdot \vec{p}_i|. \quad (14)$$

We recognize the last term as the numerator of $T(\hat{n})$ in Eq. (4). Since the denominator of $T(\hat{n})$ is independent of $\hat{n}$, this is equivalent to the definition of thrust via axis finding in Sec. II B. Thus, thrust takes the form of an induced 2-to-1 norm problem.

By duality, with $\beta^* = \infty$, thrust can alternatively be viewed as a sup-to-2 norm problem,

$$\|M^T\|_{\infty\to2} = \max_{\|s\|_\infty=1} \|sM\|_2 = \max_{s_i\in\{-1,+1\}} \left\|\sum_{i=1}^{N} s_i\vec{p}_i\right\|_2. \quad (15)$$

This corresponds to the definition of thrust via partitioning in Sec. II A, since setting $s_i = -1$ denotes flipping the orientation of vector $\vec{p}_i$ relative to the partitioning plane, while setting $s_i = 1$ retains the orientation of $\vec{p}_i$.

Therefore, we see that the problem of computing thrust in particle physics is in fact a special instance of the more general problem of computing induced matrix norms. While there exist choices of α and β for which efficient algorithms for computing $\|M\|_{\alpha\to\beta}$ exist for arbitrary M, it is believed that the general problem of computing the induced 2-to-1 norm and that of computing the induced ∞-to-2 norm are both NP hard [59–61]. This suggests that thrust is an excellent test bed to explore possible gains from quantum computation.

D. Alternative duality derivation

There is alternative language to understand this thrust duality that will be useful for the generalizations in Sec. VII. This approach is based on Ref. [62], which showed that different jet finding strategies can sometimes be derived from a common metaoptimization problem.

Consider a partition H (not necessarily defined by a plane) with total three-momentum,

$$\vec{P} = \sum_{i\in H} \vec{p}_i. \quad (16)$$

Our analysis is based on the following objective function that depends on both a choice of partition and a choice of axis:

$$O(\vec{P}, \hat{n}) = \hat{n} \cdot \vec{P} + \lambda(\hat{n}^2 - 1), \quad (17)$$

where λ is a Lagrange multiplier to enforce that the axis $\hat{n}$ has unit norm. At this point, $\vec{P}$ and $\hat{n}$ are completely independent entities, and $\hat{n}$ does not play any role in determining the partition H.

For fixed $\vec{P}$, we can optimize $O(\vec{P}, \hat{n})$ over $\hat{n}$,

$$\hat{n}_{\text{opt}} = \frac{\vec{P}}{|\vec{P}|}. \quad (18)$$

Plugging this into Eq. (17) yields

$$O(\vec{P}) \equiv O(\vec{P}, \hat{n}_{\text{opt}}) = |\vec{P}|, \quad (19)$$

which is (half) of the thrust numerator in Eq. (2).

For fixed $\hat{n}$, we can optimize $O(\vec{P}, \hat{n})$ over $\vec{P}$ (or equivalently, over the partition H),

$$\vec{P}_{\text{opt}} = \sum_{i=1}^{N} \Theta(\hat{n} \cdot \vec{p}_i)\vec{p}_i. \quad (20)$$

Plugging this into Eq. (17) yields

$$O(\hat{n}) \equiv O(\vec{P}_{\text{opt}}, \hat{n}) = \sum_{i=1}^{N} \Theta(\hat{n} \cdot \vec{p}_i)(\hat{n} \cdot \vec{p}_i), \quad (21)$$

which is (half) of the thrust numerator in Eq. (7).

Since the order of optimization is irrelevant to the final optimum, this again shows that the two thrust definitions are dual. Either way, the maximum value of the objective function will be

$$O(\vec{P}_{\text{opt}}, \hat{n}_{\text{opt}}) = |\vec{P}_{\text{opt}}|, \quad (22)$$

which, following Ref. [57], is just the maximum achievable polygon diagonal.

III. CLASSICAL ALGORITHMS

We now describe the best known classical algorithm for thrust in the literature, which requires $O(N^3)$ time, and then show how it can be improved to $O(N^2 \log N)$ using a sorting technique from Ref. [3]. We start by assuming a sequential classical computing model in this section and end with a brief discussion of parallel classical computing.

A. Plane partitioning via a reference axis

The best known classical thrust algorithm [2] uses the reference axis approach discussed at the end of Sec. II B.[1] This is the thrust algorithm implemented in PYTHIA as of

---

[1]Strangely, Ref. [2] claims $O(N^2)$ usage, which only includes the number of partitions to check, not the computation of thrust itself.

version 8 [63].[2] The key realization is that, because of Eq. (8), one only needs to search over inequivalent plane partitions. Two particles are sufficient to determine a separating plane, so there are $O(N^2)$ inequivalent plane partitions to consider. For each partition, determining $T(H_L)$ takes $O(N)$, leading to an $O(N^3)$ algorithm.

More specifically, for each pair of particles $\vec{p}_i$ and $\vec{p}_j$, one determines a reference axis $\hat{r}_{ij}$ normal to the plane spanned by them,

$$\hat{r}_{ij} \equiv \frac{\vec{p}_i \times \vec{p}_j}{|\vec{p}_i \times \vec{p}_j|}. \qquad (23)$$

Then, each particle $\vec{p}_k$ is either assigned to the hemisphere H_{ij} if $\hat{r}_{ij} \cdot \vec{p}_k > 0$ or ignored if $\hat{r}_{ij} \cdot \vec{p}_k < 0$. Cases where $\hat{r}_{ij} \cdot \vec{p}_k = 0$ are ambiguous, and we provide a general strategy to deal with this in Sec. III B below. At minimum, we have to treat the cases where $k = i$ or j, which requires testing $2 \times 2 = 4$ possibilities for whether or not $\vec{p}_i$ and/or $\vec{p}_j$ should be included in H_{ij}, for a total of $4N(N-1)$ partitions. (This can be reduced by a factor of 2 using momentum conservation, since $\hat{r}_{ij}$ and $\hat{r}_{ji}$ define the same hemispheres.) The final hemisphere jets are determined by the partition that maximizes

$$T_{ij} \equiv T(H_{ij}). \qquad (24)$$

Note that, in general, none of the $O(N^2)$ reference axes considered will align with the actual thrust axis. Nevertheless, the *partitions* defined by $\hat{r}_{\mathrm{opt}}$ and $\hat{n}_{\mathrm{opt}}$ will be identical. (In the idealized case of infinitesimal radiation everywhere in the event, all possible separating planes would be considered, so $\hat{r}_{\mathrm{opt}}$ would then equal $\hat{n}_{\mathrm{opt}}$.) Once the optimal partition is known, the thrust axis itself is determined by Eq. (8).

In terms of computational complexity, for a fixed hemisphere H_{ij}, it takes $O(N)$ time to compute the hemisphere three-momentum in Eq. (16). The thrust denominator $T_{\mathrm{denom}} = \sum_{i=1}^{N} |\vec{p}_i|$ also takes $O(N)$ time, but it can be precomputed since it is independent of the partition. Once $\vec{P}_{ij}$ and T_{denom} are known, though, it only takes $O(1)$ time to determine the value of T_{ij},

$$T_{ij} = \frac{2|\vec{P}_{ij}|}{T_{\mathrm{denom}}}, \qquad (25)$$

where we used Eq. (8) to derive this expression. For the best known classical algorithm, there are $O(N^2)$ partitions, and we have to do an $O(N)$ computation of $T(H_{ij})$ for each

partition, leading to the $O(N^3)$ scaling. In Sec. III C, we can improve on this runtime by iteratively updating $\vec{P}_{ij}$ in a special order.

B. Doubling trick

To simplify the thrust algorithm, it is convenient to artificially double the number of particles. Starting from N three-momenta, we create a list of length $2N$ by including both $\vec{p}_k$ and its negative $-\vec{p}_k$. Because $\vec{p}_k$ and $-\vec{p}_k$ can never be in the same hemisphere, and because of the momentum conservation relation in Eq. (2), this doubling trick has no effect on the value of thrust. It does, however, provide us with a convenient way to deal with the fourfold ambiguity above, since we can now define the hemisphere H_{ij} to always include particle i and particle j.

To deal with cases where $\hat{r}_{ij} \cdot \vec{p}_k = 0$ (i.e., any time three or more particles are coplanar), we offset the reference axis by

$$\hat{r}_{ij} \to \hat{r}_{ij} + \epsilon \vec{q}_{ij}, \qquad \vec{q}_{ij} \equiv \frac{\vec{p}_i}{|\vec{p}_i|} + \frac{\vec{p}_j}{|\vec{p}_j|}, \qquad (26)$$

and then take the formal $\epsilon \to 0$ limit. Specifically, if $\hat{r}_{ij} \cdot \vec{p}_k = 0$, then particle $\vec{p}_k$ is included in H_{ij} if $\vec{q}_{ij} \cdot \vec{p}_k > 0$ and ignored otherwise.

Crucially, Eq. (26) ensures that $\vec{p}_i$ and $\vec{p}_j$ are always in the hemisphere H_{ij}, but $-\vec{p}_i$ and $-\vec{p}_j$ are not. (One has to be mindful of the pathological situation where $\vec{p}_i$ and $\vec{p}_j$ are exactly antiparallel, though in this case, thrust is determined by one of the other hemisphere partitions.) The hemisphere three-momentum is now

$$\vec{P}_{ij} = \frac{1}{2} \sum_{k \in H_{ij}} \vec{p}_k, \qquad (27)$$

where the factor of $\frac{1}{2}$ compensates for the artificial doubling.

We will use this doubling trick repeatedly in this paper, though not for quantum annealing in Sec. IV where it is counterproductive. To simplify the description of the algorithms, we will leave implicit the treatment of all $\hat{r}_{ij} \cdot \vec{p}_k = 0$ cases via Eq. (26). It is worth mentioning that an alternative way to deal with coplanar configurations is to offset the momenta by a small random amount, but we find the doubling trick to be more convenient in practice since it avoids the fourfold ambiguity automatically.

C. Improvements via sort

The $O(N^3)$ algorithm can be further improved to run in time $O(N^2 \log N)$.[3] This can be achieved using a strategy from SISCone [3] which uses a clever choice of traversal order. Note that SISCone is intended for proton-proton

---

[2]Version 6 of PYTHIA [64] uses a heuristic to approximate thrust, via an iterative procedure that updates the partition starting from seed axes. While this method converges very quickly, it only finds a local maximum, not the global one [57], though this may be sufficient for practical applications. See related discussion in Ref. [32].

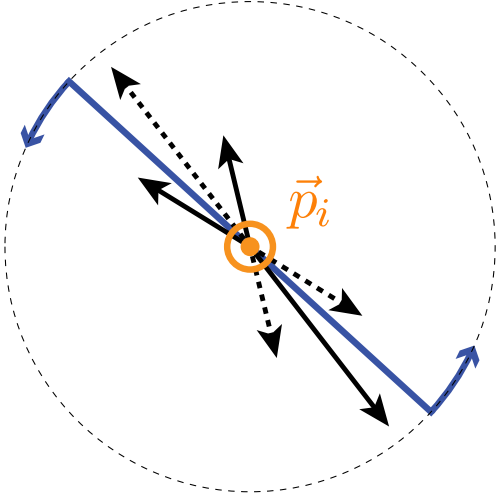[3]We thank Gregory Soyez for discussions related to this point.

FIG. 2. Illustration of the sorting algorithm around the $\vec{p}_i$ axis (seen from the top down). The dashed vectors correspond to the doubling trick. As the blue partitioning plane sweeps in azimuth, the hemisphere momentum is updated according to Eq. (30).

collisions, whereas our interest here is in electron-positron collisions, but the same basic strategy still applies.

The goal of SISCone is to find conical jet configurations J where the enclosed particles are within a distance R from the cone axis $\hat{n}_J$. Moreover, these cone jets must be *stable*, meaning that the jet three-momentum $\vec{P}_J = \sum_{i \in J} \vec{p}_i$ is aligned with the cone axis $\hat{n}_J$. Like thrust, SISCone involves solving a partitioning problem where the naive brute-force approach requires $O(N2^N)$ time. Like for thrust, one can reduce the naive runtime for SISCone to $O(N^3)$ using the fact that two points lying on the circumference of a circle are sufficient to determine the cone constituents. There is an eightfold ambiguity in the cone assignments, which we discuss further in Sec. VII C.

The key insight of Ref. [3] is that one need not recompute $\vec{P}_J$ for all $O(N^2)$ candidate cones. Ignoring the eightfold ambiguity, let the candidate cones be labeled by i and j. For fixed i, one can define a special traversal order for j such that only one particle enters or leaves the cone at a time. There are N particles labeled by i, and for fixed i, sorting over j takes $O(N \log N)$ time. After the initial $O(N)$ determination of $\vec{P}_J$ for the first j values in the sorted list, updating the value of $\vec{P}_J$ for each j iteration only requires $O(1)$ time, since you need to only add the momentum of a point entering the cone or subtract the momentum of a point leaving the cone. Thus, the final algorithm is $O(N^2 \log N)$.

We can apply exactly the same sorting strategy to the computation of thrust, as shown in Fig. 2. The reason is that the reference axis $\hat{r}_{ij}$ depends only on the cross product $\vec{p}_i \times \vec{p}_j$. This means that for fixed $\vec{p}_i$, we can choose an ordering of the $\vec{p}_j$ such that the partitions induced by $\{\hat{r}_{i1}, ..., \hat{r}_{iN}\}$ are specified by a single sortable parameter.

To see this, it is convenient to transform to a coordinate system where $\vec{p}_i$ points in the z direction, i.e., $\vec{p}_i = |\vec{p}_i|(0,0,1)$. For any $j \neq i$, we can write $\vec{p}_j$ in spherical coordinates as $\vec{p}_j = |\vec{p}_j|(\sin\theta_j \cos\phi_j, \sin\theta_j \sin\phi_j, \cos\theta_j)$, where θ_j is the polar angle and ϕ_j is the azimuthal angle. Then $\hat{r}_{ij} = (-\sin\phi_j, \cos\phi_j, 0)$, so the partition is indeed determined by the single parameter ϕ_j, independent of θ_j. Specifically, particle k is in hemisphere H_{ij} if

$$\hat{r}_{ij} \cdot \hat{p}_k = \sin\theta_k \sin(\phi_k - \phi_j) \qquad (28)$$

is positive. This implies that $0 < \phi_k - \phi_j < \pi$, where azimuthal angle differences are calculated modulo 2π.

Furthermore, because of the doubling trick, there is a simple way to determine which particles are in the partition. With the doubling, there are $2N$ possible choices for i, and by Eq. (26) we know that the doubler $-\vec{p}_i$ cannot be in the same partition as $\vec{p}_i$. Using the above coordinate system, we can sort the remaining $2N - 2$ vectors according to their ϕ coordinates, so that $0 \leq \phi_{j_1} \leq \phi_{j_2} \leq \cdots \leq \phi_{j_{2N-2}} < 2\pi$. (In cases where two particles happen to have identical values of ϕ_j, their relative ordering does not matter for the argument below, as long as the doublers are also put in the same order.) Crucially, for a particle at position a in this sorted list, its doubler (which is π away in azimuth) must be at position $a + N - 1$. To see why, note that a hemisphere either contains a particle or contains its doubler, so there must be exactly N particles in each hemisphere. Particle i is already accounted for, meaning that any candidate partition must contain $N - 1$ entries from the sorted list. Since the sorted list is ordered by azimuth, and since the partitioning is determined by azimuth alone via Eq. (28), the $N - 1$ elements from position a to position $a + N - 2$ inclusive must be in a common partition, and the doubler must be the next one on the list. Therefore, candidate thrust partitions always take the form

$$H_{i,j_a} = \{i, j_a, j_{a+1}, ..., j_{a+N-2}\}. \qquad (29)$$

(Note that, as in Eq. (26), both particle i and particle j_a are always contained in H_{i,j_a}.)

These observations allow us to construct an $O(N^2 \log N)$ algorithm for thrust. The outer loop involves iterating over all $2N$ choices for i. The inner loop involves the following $O(N \log N)$ algorithm. We perform the sorting procedure above for fixed i, which takes $O(N \log N)$ time. For the first element in the sorted list, we determine the partition H_{i,j_1} using Eq. (29) with $a = 1$. We can readily compute $\vec{P}_{i,j_1}$ via Eq. (27) in time $O(N)$ and then compute the associated thrust value via Eq. (25) in $O(1)$. For the subsequent $2N - 3$ elements of the sorted list, we step through them one by one, updating the partition from $H_{i,j_a} = \{i, j_a, j_{a+1}, ..., j_{a+N-2}\}$ to $H_{i,j_{a+1}} = \{i, j_{a+1}, j_{a+2}, ..., j_{a+N-1}\}$. In doing so, we need

to subtract $\vec{p}_{j_a}$ and add $\vec{p}_{j_{a+N-1}}$ (which is the same as $-\vec{p}_{j_a}$ by the doubling trick), leading to the update step,

$$\vec{P}_{i,j_{a+1}} = \vec{P}_{i,j_a} - \vec{p}_{j_a}, \tag{30}$$

where one has to remember the factor of $\frac{1}{2}$ in Eq. (27). From the updated momentum, we recompute the associated thrust value via Eq. (25) in $O(1)$ time. The total time from stepping through the $2N - 3$ partition momenta is $O(N)$, so the inner loop is dominated just by the initial $O(N \log N)$ sorting step. The maximum T_{ij} over all i and sorted j determines the final hemisphere jets.

D. Parallel classical algorithm

The sorting algorithm above requires $O(N^2 \log N)$ operations. In a model with a single CPU and random-access memory, this corresponds to time $O(N^2 \log N)$ as well. We can also consider parallel computing models in which the N words of memory are accompanied by N parallel processors; see Ref. [65] for more discussion of these models. In this case, we will see that a runtime of $O(N \log N)$ can be achieved. For simplicity, we do not consider the general case in which the number of parallel CPUs and the amount of memory can be varied independently, nor will we discuss the varying models of parallel computing in Ref. [65].

We briefly sketch here how the sorting strategy in Sec. III C can be sped up with parallel processors. There are three main computational bottlenecks: iterating over all particles i (C_{iter}), sorting over particles j for fixed i (C_{sort}), and determining the hemisphere constituents over each j for fixed i (C_{hemi}), leading to a runtime of $O(C_{\text{iter}}(C_{\text{sort}} + C_{\text{hemi}}))$. For sequential classical computing, we found $C_{\text{iter}} = O(N)$, $C_{\text{sort}} = O(N \log N)$, and $C_{\text{hemi}} = O(N)$. A parallel computer cannot improve on C_{iter}, but there are parallel computing algorithms for sorting [66] and partial sums [67] that would allow us to achieve $C_{\text{sort}} = C_{\text{hemi}} = O(\log N)$, leading to a $O(N \log N)$ runtime. We will compare the quantum and classical parallel architectures in Sec. VI.

IV. THRUST VIA QUANTUM ANNEALING

The first quantum algorithm we describe is based on quantum annealing [6,7]. In a quantum annealer such as the D-Wave system [8], the solution to an optimization problem is encoded in the ground state of a target Hamiltonian. Such a Hamiltonian takes the form of an Ising model,

$$H(\{s_i\}) = \sum_{i=1}^{N} h_i s_i + \sum_{i<j=1}^{N} J_{ij} s_i s_j, \tag{31}$$

where each of the N Ising spins $s_i \in \{-1, +1\}$ corresponds to a qubit, and the $\{h_i\}$ and $\{J_{ij}\}$ correspond to programmable weights and couplings between qubits, respectively.

Equivalently, under the transformation $s_i = 2x_i - 1$, we can frame the optimization problem as a QUBO problem, where the objective function takes the form

$$O(\{x_i\}) = \sum_{i,j=1}^{N} Q_{ij} x_i x_j \tag{32}$$

for $x_i \in \{0, 1\}$. Note that the fact that i, j are now summed with repeated indices and the fact that $x_i^2 = x_i$ allow us to absorb the linear terms into the quadratic terms.

For the thrust problem, it is convenient to first define the three-momentum of a candidate partition as

$$\vec{P}(\{x_i\}) = \sum_{i=1}^{N} \vec{p}_i x_i, \tag{33}$$

where $x_i = 1$ if particle $\vec{p}_i$ is in the partition and $x_i = 0$ otherwise. Following Eq. (25), the thrust of this partition is given by

$$T(\{x_i\}) = \frac{2|\vec{P}|}{T_{\text{denom}}} = \frac{2}{T_{\text{denom}}} \sqrt{\sum_{i,j=1}^{N} \vec{p}_i \cdot \vec{p}_j x_i x_j}. \tag{34}$$

Because of the square root factor, this is not a QUBO problem, but since the optimal partition is the same for any monotonic rescaling of $T(\{x_i\})$, we can optimize the squared relation

$$T(\{x_i\})^2 = \frac{4}{T_{\text{denom}}^2} \sum_{i,j=1}^{N} \vec{p}_i \cdot \vec{p}_j x_i x_j, \tag{35}$$

which now takes the form of the QUBO problem in Eq. (32), as desired. Finding the ground state of $-T(\{x_i\})^2$ (note the minus sign) is the same as determining thrust.

The space usage of a quantum annealing algorithm is $O(N)$, corresponding to one qubit for each x_i. The annealing time required depends on the spectral gap of the particular Hamiltonian, and we leave the question of determining the spectral gap of the thrust objective function to future work.

V. THRUST VIA QUANTUM SEARCH

We now describe a quantum algorithm for thrust based on Grover search. We first describe the algorithm in terms of two abstract operations, LOOKUP and SUM, both of which perform data loading in superposition. Then, we describe two computing models for loading the classical data into quantum memory: the sequential model and the parallel model. Key to the algorithmic speedups we achieve is the fact that even if quantum data loading takes time $O(N)$, other calculations inside the Grover search loop also

1. Randomly pick an index j and set `curr_max = j`.

2. Set `iter_count = 0` and `max_it = 1`.

3. While `iter_count` $< O(\sqrt{K})$:

   (a) While `max_it` $< O(\sqrt{K})$:

      i. Prepare the initial state $|\psi_0\rangle = \frac{1}{\sqrt{K}} \sum_{i=1}^{K} |i\rangle$.

      ii. Choose `grov_steps` uniformly at random from $\{0, 1, ..., \text{max\_it} - 1\}$.

      iii. Set `iter_count = iter_count + grov_steps`.

      iv. Repeat `grov_step` times:

         A. Reflect about states satisfying $A[i] > A[\text{curr\_max}]$;

         B. Reflect about the initial state $|\psi_0\rangle$.

      v. Measure the first register to obtain index j; if $A[j] > A[\text{curr\_max}]$, set `curr_max = j` and break.

   (b) Let `max_it` $= \mu \times$ `max_it`, where μ is a constant between 1 and 4/3.

4. Measure the first register to obtain index j; if $A[j] > A[\text{curr\_max}]$, set `curr_max = j`.

FIG. 3.   Quantum search algorithm due to Dürr-Høyer to find the index corresponding to the maximum entry of an array $A[i]$ with K elements [11]. The number of Grover steps is chosen at random, since this is a search over an unknown number of marked times [10].

take $O(N)$ in both the classical and quantum models, so we gain from decreasing the effective search space from $O(N^2)$ to $O(N)$. The sequential model results in an algorithm that requires $O(N^2)$ time and $O(\log N)$ qubits. The parallel model requires $O(N \log N)$ time and $O(N \log N)$ qubits. We also assess how the resource requirements of these algorithms scale with the precision of the computation.

A. Algorithm overview

Our quantum thrust algorithm is based on the quantum maximum finding algorithm of Dürr and Høyer [11], which returns the maximum element of an unsorted array with K elements in $O(\sqrt{K})$ time, assuming quantum query access to the array. This algorithm is itself a generalization of Grover search [9].

In this context, quantum query access means that for an array $A[1], ..., A[K]$, we can efficiently perform a unitary operation U such that

$$U_A |i\rangle |0\rangle = |i\rangle |A[i]\rangle, \tag{36}$$

along with its inverse $U^\dagger$. The first register, containing $|i\rangle$, should have dimension at least K, so that $|1\rangle, ..., |K\rangle$ are each orthogonal states of the register, and the second register should be large enough to store the values $A[i]$. Note that Eq. (36) does not fully specify the unitary U_A since it does not specify its action when the second register is not initially in the state $|0\rangle$. One possible way to define U_A fully is to have $U|i\rangle |x\rangle = |i\rangle |x + A[i]\rangle$ with addition defined over an appropriately sized finite ring such as $\mathbb{Z}_2^n$, but this is not necessary for applications such as in Refs. [9,11]. Quantum query access to an array A is more

demanding than simply having A stored on disk, as we will discuss below.

Recall that Grover search finds one marked item out of an array of K items, assuming the ability to reflect about the marked item. Reference [10] further extends Ref. [9] to find one marked item when there are $t > 1$ marked items, assuming the ability to reflect about the multiple marked items. Generic Grover search then consists of the following steps:

   (1) Prepare the initial state $|\psi_0\rangle = \frac{1}{\sqrt{K}} \sum_{i=1}^{K} |i\rangle$.
   (2) Repeat $O(\sqrt{K/t})$ times:
      (a) Reflect about the marked states.
      (b) Reflect about the initial state $|\psi_0\rangle$.

When the number t of marked items is unknown, Ref. [10] employs an exponential searching algorithm that guesses the number of marked items, increasing the guess by a constant factor each time. This is a probabilistic algorithm that performs a measurement for each guess, finding a solution in overall expected time $O(\sqrt{K/t})$.

The maximum finding algorithm of Ref. [11], summarized in Fig. 3, is based on this probabilistic exponential searching algorithm. It keeps track of the current best maximum seen so far and considers marked states to be those that have a larger array entry value than the current maximum. It employs the Grover-based exponential searching algorithm of Ref. [10] for an unknown number of marked states, performing measurements to obtain the maximum with probability at least $1/2$. If desired, we can improve the success probability to $1 - \eta$ with $\eta > 0$, at the cost of an extra $O(\log 1/\eta)$ factor, by performing $O(\log 1/\eta)$ rounds of the algorithm.

Our quantum thrust algorithms are then a direct application of quantum maximum finding, but now to an array with $K = O(N^2)$ entries corresponding to the choice of separating plane. To deal with the fourfold ambiguity, we

use the doubling trick of Sec. III C, including each original vector $\vec{p}_k$ and its negative $-\vec{p}_k$ in the list of three-momenta to obtain a search space of size $K = 4N^2$. Our problem, now, is to find the maximum value of T_{ij} with i and j each ranging over $2N$ possible indices. This requires us to be able to load the momentum vectors corresponding to each array index, which means that the quantum algorithm must have some means of accessing the classical data.

B. Data loading considerations

We can describe our quantum thrust algorithms in terms of two abstract operations, LOOKUP and SUM. Their implementation will be described in Sec. V C for the sequential model and Sec. V E for the parallel model. Beyond thrust, these operations are quite general in their application to loading classical data into quantum algorithms.

Note that our search space is of size $O(N^2)$, while data loading over N items in a classical database takes time $O(N)$. Therefore, we can conceptualize our quantum speedup as resulting from being able to perform data loading over the *superposition* of search space items. It is important here that the set of search space items is not the same as the set of data points. In general, any application of Grover search over a search space of size $O(N^\alpha)$ with $\alpha \geq 2$ will result in a square root speedup, whereas for $\alpha < 2$, the cost of the algorithm will be dominated by the $O(N)$ data loading cost.

The LOOKUP operation is queried with one index corresponding to a given particle, returning the momentum corresponding to that index,

$$U_{\text{LOOKUP}}|i\rangle|\vec{0}\rangle = |i\rangle|\vec{p}_i\rangle. \tag{37}$$

Note that the second register, initialized as $|\vec{0}\rangle$, has to be large enough to store the three-momenta to the desired (qu)bit accuracy. To make U_{LOOKUP} unitary, we define $U_{\text{LOOKUP}}|i\rangle|\vec{q}\rangle = |i\rangle|\vec{q} + \vec{p}_i\rangle$ for general vectors $\vec{q}$, where the addition is done modulo some value larger than the maximum momentum encountered in the problem. To deal with pairs of particles, we can call U_{LOOKUP} twice on different registers to map $|i\rangle|j\rangle|\vec{0}\rangle|\vec{0}\rangle \rightarrow |i\rangle|j\rangle|\vec{p}_i\rangle|\vec{p}_j\rangle$. This LOOKUP operation will be used to determine all $O(N^2)$ reference axes $\hat{r}_{ij}$, taking $O(N)$ time in the sequential model and $O(\log N)$ time in the parallel model.

The SUM operation returns the sum over all momenta, possibly with a transformation $f(\vec{p}; c)$ applied to each momentum vector,

$$U_{\text{SUM}}|c\rangle|0\rangle = |c\rangle|\sum_{k=1}^{N} f(\vec{p}_k; c)\rangle, \tag{38}$$

where c represents possible control qubits. From a given reference axis $\hat{r}_{ij}$, SUM will be used to calculate the value of T_{ij}. It is crucial that calculating T_{ij} for fixed i and j takes the same runtime as LOOKUP, i.e., $O(N)$ for sequential and

$O(\log N)$ for parallel. Notably, a wide class of collider observables can be computed in linear runtime [68], even those that would naively scale like a high polynomial power.

Using LOOKUP and SUM, our quantum thrust algorithm is described in Fig. 4. As with standard Grover search, we need to be able to reflect about the initial state and the marked states, namely, those whose corresponding values of thrust are larger than the best maximum seen so far. To identify the marked states, we compute thrust for each choice of separating plane, using LOOKUP and SUM to interface the quantum algorithm with the classical data. We uncompute intermediate steps of our calculations using standard methods (e.g., Sec. 3.2 of Ref. [4]) to make sure that, after computing T_{ij}, the system can be reflected about the initial state.

Let C_{LOOKUP} be the asymptotic cost of LOOKUP and C_{SUM} be the asymptotic cost of SUM. The runtime of this algorithm is $O(N(C_{\text{LOOKUP}} + C_{\text{SUM}}))$ since there is an $O(N)$ outer Grover search loop, while the inner loop is dominated by one application of LOOKUP and one application of SUM. Note that the computation of the initial guess for the maximum, T_{mn}, can be performed in $O(N)$ time classically, while preparation of the initial state and reflection about the initial state can each be performed in $O(\log N)$ time, the time required to perform a Hadamard gate.

C. Sequential computing model

The first computing model we consider is one in which one gate, classical or quantum, can be executed per time step. We should think of the classical computer as controlling the overall computation. In a single time step, it can either (a) perform a classical logic gate, (b) choose a quantum gate or measurement, or (c) read a word from the input (e.g., a single momentum). Another way to think about this model is that we measure cost by the circuit size, i.e., the total number of gates.

While fault-tolerant quantum computers are expected to require parallel control to perform error correction, there are still plausible models in which the cost of the computation will be proportional to the number of logical gates. One possibility is that the cost is dominated by generating magic states or by long-range interactions. Another possibility is that we are using a small quantum computer without fault tolerance, but in an architecture such as a one-dimensional ion trap, where the available gates are long range and cannot be parallelized.

Under this sequential model, the operations LOOKUP and SUM each take $O(N)$ time and require $O(\log N)$ qubits. Specifically, LOOKUP requires a register of size $O(\log N)$ to store the query index i, along with a register to store the requested three-momentum $\vec{p}_i$. It operates by performing a sequential scan through all N items in the classical database to fetch and return $\vec{p}_i$. More concretely, in $O(1)$ time, we can perform $U_{\text{LOOKUP},i}$, defined by

1. Randomly pick indices m, n and set `curr_max` $= (m, n)$.

2. Compute `p_sum` $= \frac{1}{2} \sum_{i=1}^{2N} |\vec{p}_i|$.

3. Set `iter_count` $= 0$ and `max_it` $= 1$.

4. While `iter_count` $< O(N)$:

    (a) While `max_it` $< O(N)$:

        i. Prepare the initial state $|\psi_0\rangle = \frac{1}{2N} \sum_{i,j=1}^{2N} |i\rangle |j\rangle |\vec{0}\rangle |\vec{0}\rangle |\hat{0}\rangle |\vec{0}\rangle |0\rangle$.

        ii. Choose `grov_steps` uniformly at random from $\{0, 1, ..., \texttt{max\_it} - 1\}$.

        iii. Let `iter_count` $=$ `iter_count` $+$ `grov_steps`.

        iv. Repeat `grov_steps` times:

            A. Call subroutine `COMP_T` to compute T_{ij}:

$$|i\rangle |j\rangle |\vec{0}\rangle |\vec{0}\rangle |\hat{0}\rangle |\vec{0}\rangle |0\rangle \mapsto |i\rangle |j\rangle |\vec{0}\rangle |\vec{0}\rangle |\hat{0}\rangle |\vec{0}\rangle |T_{ij}\rangle.$$

            B. Reflect about states with $T_{ij} > T_{\texttt{curr\_max}}$ with a phase factor:

$$|i\rangle |j\rangle |\vec{0}\rangle |\vec{0}\rangle |\hat{0}\rangle |\vec{0}\rangle |T_{ij}\rangle \mapsto (-1)^{\Theta(T_{ij} - T_{\texttt{curr\_max}})} |i\rangle |j\rangle |\vec{0}\rangle |\vec{0}\rangle |\hat{0}\rangle |\vec{0}\rangle |T_{ij}\rangle.$$

            C. Uncompute the T_{ij} register to obtain state: $(-1)^{\Theta(T_{ij} - T_{\texttt{curr\_max}})} |i\rangle |j\rangle |\vec{0}\rangle |\vec{0}\rangle |\hat{0}\rangle |\vec{0}\rangle |0\rangle$.

            D. Reflect about the initial state using $R_0 = 2 |\psi_0\rangle \langle\psi_0| - I^{\otimes 7}$.

        v. Measure the $\{i, j, T_{ij}\}$ registers to obtain $\{k, \ell, T_{k\ell}\}$; if $T_{k\ell} > T_{\texttt{curr\_max}}$, set `curr_max` $= (k, \ell)$ and break.

    (b) Let `max_it` $= \mu \times$ `max_it`, where μ is a constant between 1 and 4/3.

5. Measure the $\{i, j, T_{ij}\}$ registers to obtain $\{k, \ell, T_{k\ell}\}$; if $T_{k\ell} > T_{\texttt{curr\_max}}$, set `curr_max` $= (k, \ell)$.

Subroutine COMP_T:

1. Load $\vec{p}_i, \vec{p}_j$ using LOOKUP:

$$|i\rangle |j\rangle |\vec{0}\rangle |\vec{0}\rangle |\hat{0}\rangle |\vec{0}\rangle |0\rangle \mapsto |i\rangle |j\rangle |\vec{p}_i\rangle |\vec{p}_j\rangle |\hat{0}\rangle |\vec{0}\rangle |0\rangle.$$

2. Calculate the reference axis via $\hat{r}_{ij} = (\vec{p}_i \times \vec{p}_j)/|\vec{p}_i \times \vec{p}_j|$:

$$|i\rangle |j\rangle |\vec{p}_i\rangle |\vec{p}_j\rangle |\hat{0}\rangle |\vec{0}\rangle |0\rangle \mapsto |i\rangle |j\rangle |\vec{p}_i\rangle |\vec{p}_j\rangle |\hat{r}_{ij}\rangle |\vec{0}\rangle |0\rangle.$$

3. Apply SUM, with $f(\vec{p}_k; \hat{r}_{ij}) = \{\vec{p}_k/2 \text{ if } \hat{r}_{ij} \cdot \vec{p}_k > 0; \vec{0} \text{ if } \hat{r}_{ij} \cdot \vec{p}_k < 0\}$, to obtain hemisphere momentum $\vec{P}_{ij}$:

$$|i\rangle |j\rangle |\vec{p}_i\rangle |\vec{p}_j\rangle |\hat{r}_{ij}\rangle |\vec{0}\rangle |0\rangle \mapsto |i\rangle |j\rangle |\vec{p}_i\rangle |\vec{p}_j\rangle |\hat{r}_{ij}\rangle |\vec{P}_{ij}\rangle |0\rangle.$$

4. Calculate thrust via $T_{ij} = 2|\vec{P}_{ij}|/\texttt{p\_sum}$:

$$|i\rangle |j\rangle |\vec{p}_i\rangle |\vec{p}_j\rangle |\hat{r}_{ij}\rangle |\vec{P}_{ij}\rangle |0\rangle \mapsto |i\rangle |j\rangle |\vec{p}_i\rangle |\vec{p}_j\rangle |\hat{r}_{ij}\rangle |\vec{P}_{ij}\rangle |T_{ij}\rangle.$$

5. Uncompute registers to obtain state: $|i\rangle |j\rangle |\vec{0}\rangle |\vec{0}\rangle |\hat{0}\rangle |\vec{0}\rangle |T_{ij}\rangle$.

FIG. 4. Our Grover-based quantum thrust algorithm, written in terms of the abstract LOOKUP and SUM operations. The symbols $|\vec{0}\rangle$, $|\hat{0}\rangle$, and $|0\rangle$ refer to initial states for a three-momentum, normalized axis, and real number, respectively. Note that we have applied the doubling trick from Sec. III B, such that each $\vec{p}_k$ has its negative $-\vec{p}_k$ in the set of three-momenta. Cases where $\hat{r}_{ij} \cdot \vec{p}_k = 0$ are treated implicitly via Eq. (26). A key difference compared to Fig. 3 is that the quantity to maximize, T_{ij}, is calculated quantumly via the COMP_T subroutine.

$$U_{\text{LOOKUP},i} |i\rangle |\overrightarrow{0}\rangle = |i\rangle |\overrightarrow{p_i}\rangle, \tag{39a}$$

$$U_{\text{LOOKUP},i} |j\rangle |\overrightarrow{0}\rangle = |i\rangle |\overrightarrow{0}\rangle, \qquad \text{if } j \neq i. \tag{39b}$$

Then we implement U_{LOOKUP} in Eq. (37) by performing $U_{\text{LOOKUP},1} U_{\text{LOOKUP},2} \cdots U_{\text{LOOKUP},N}$ in time $O(N)$. Similarly,

SUM takes time $O(N)$ because it also performs one pass through all N items in the classical database while computing and returning the sum $\sum_{i=1}^{N} f(\vec{p}_i; c)$.

With this implementation of LOOKUP and SUM, with $C_{\text{LOOKUP}} = C_{\text{SUM}} = O(N)$, the Grover-search based thrust algorithm in Fig. 4 requires $O(N^2)$ time and $O(\log N)$ qubits.

D. Quantum improvements via sort?

One might wonder whether the runtime of the quantum thrust algorithm could be reduced from $O(N^2)$ to $O(N^{3/2} \log N)$, using the same strategy that we used in Sec. III C to reduce the classical thrust algorithm time from $O(N^3)$ to $O(N^2 \log N)$. The answer is yes, in principle, but it would require a computing model beyond the sequential one.

Recall that two points define the partitioning plane, and after selecting the first point, we could sort the second point according to a special traversal order. This allowed us to avoid the $O(N)$ cost of resumming the momenta for each candidate plane. Quantum algorithms require $\Omega(N \log N)$ time for sort [69], which means that they cannot be used to speed up this part of the classical algorithm. In principle, though, we could still obtain a Grover square root speedup when searching over the $O(N)$ candidates for the first point determining the partitioning plane. Combining the $O(\sqrt{N})$ Grover search over the first point with the $O(N \log N)$ sort over the second point would then yield an $O(N^{3/2} \log N)$ overall algorithm.

The challenge here is that to perform quantum sort, all of the data need to be stored somehow in quantum memory, which goes beyond the sequential computing model above where only one data point is ever accessed in a given time step. We leave to future work the design of a quantum computing architecture suitable for loading and sorting data from a classical database.

Assuming that such a sort-friendly architecture exists, one might ask about the origin of the $O(N^2 \log N)$ to $O(N^{3/2} \log N)$ speed up. Such an improvement is only possible since the strategy in Sec. III C converts thrust into a *structured search* problem [70,71], which evades the naive bounds on quantum search performance. Of course, no matter the degree of structure, we can never do better than the $O(N)$ cost to examine each data point once.

E. Parallel computing model

The parallel computing model reduces the time usage of the sequential model at the expense of additional space usage.[4] Under this model, the operations LOOKUP and SUM each take $O(\log N)$ time but require $O(N \log N)$ qubits.

An abstract version of this model is the standard quantum circuit model, in which on N qubits we can perform up to $N/2$ two-qubit gates on as many disjoint pairs of qubits as we like. A controlling classical computer with the same parallelism can also be used to process the measurement outcomes and feed the results back in to the quantum computer. To implement this in an actual quantum computer, we would need to assume long-range

---

[4]We thank Iordanis Kerenidis for discussions related to this point.

connectivity but not all-to-all connectivity. For example, Brierley [72] describes how connecting each qubit to four other qubits is enough to simulate full connectivity with $O(\log N)$ time overhead. In what follows, we neglect any $O(\log N)$ or other factors from converting the abstract circuit model to a concrete architecture.

Parallel data retrieval requires first preloading all N database items into the $O(N)$ qubits. This can be done in $O(1)$ time, since it requires only parallel copy (or CNOT) operations from the classical bits onto the qubits. (Even a cost of $O(N)$ at this stage would not change the asymptotic runtime, so one could also consider input models in which the data could only be accessed sequentially, such as tape storage.) This results in the state

$$|1\rangle|\vec{0}\rangle|2\rangle|\vec{0}\rangle\ldots|N\rangle|\vec{0}\rangle \mapsto |1\rangle|\vec{p}_1\rangle|2\rangle|\vec{p}_2\rangle\ldots|N\rangle|\vec{p}_N\rangle. \quad (40)$$

Note that this is not the same as qRAM [5], since we are loading the classical data into a product state once, and not assuming any kind of query access to the data.

Now, given our preloaded data, we can perform LOOKUP in time $O(\log N)$ by performing binary search on the query index i to locate qubits $|i\rangle|p_i\rangle$. The binary search can be made unitary using a series of $O(N)$ SWAP gates. Letting $i = i_1 i_2 \ldots i_M$ in binary, if $i_1 = 1$ we swap the first $N/2$ (i, p_i) pairs with the last $N/2$ (i, p_i) pairs, if $i_2 = 1$ we swap the first $N/4$ (i, p_i) pairs with the next $N/4$ (i, p_i) pairs, and so on. After $O(\log N)$ swaps, we end up with qubits $|i\rangle|\vec{p}_i\rangle$ in the first position. We can then copy $|\vec{p}_i\rangle$ into a blank register and uncompute the swaps.

Similarly, we can perform SUM in time $O(\log N)$ by combining the entries level by level up a binary search tree indexed by i, with $O(N)$ additional registers to store the intermediate steps. That is, we first add all pairs of entries corresponding to indices i, i' where $i_1 = i'_1, i_2 = i'_2, \ldots,$ $i_{M-1} = i'_{M-1}$ and $i_M \neq i'_M$. Then we have $N/2$ entries indexed by $j = j_1 j_2 \ldots j_{M-1}$, and again we add all pairs of entries corresponding to indices j, j' where $j_1 = j'_1, j_2 = j'_2, \ldots, j_{M-2} = j'_{M-2}$ and $j_{M-1} \neq j'_{M-1}$. Repeating this process $O(\log N)$ times allows us to sum all the entries in parallel.

Thus, the quantum thrust algorithm for the parallel data loading model, with $C_{\text{LOOKUP}} = C_{\text{SUM}} = O(\log N)$, requires $O(N \log N)$ time and $O(N \log N)$ qubits.

F. Resource requirements

In the above discussion, we focused on the scaling of our Grover-based quantum thrust algorithm in terms of the number of particles N. Here, we want to provide more information on the practical resource requirements for this algorithm in terms of the required precision of the thrust computation.

Thus far, we have been working with data in the form of three-vectors $\vec{p}_i$, where we assumed that the register holding $\vec{p}_i$ is of constant size. Just how large is this constant, given that using a finite number of qubits would

result in digitization error? For typical collider physics applications, such as anticipated for a future e^+e^- collider, we would want a dynamic range on momenta from the MeV scale (i.e., per-mille accuracy on GeV-scale hadrons) to the TeV scale (i.e., the rough energy scale for CLIC), or around 6 orders of magnitude. This means $b = \log_2 10^6 \approx$ 20 bits of accuracy. Since we are keeping track of $d = 3$ dimensions, the register holding the $\vec{p}_i$ must be of size $db = 3b$. Thus, the total number of qubits required is $O(\log_2 N + db)$ for the sequential algorithm and $O(N(\log_2 N + db))$ for the parallel algorithm.

To be more specific, the sequential version of the algorithm in Fig. 4 requires two registers with $\log_2 N$ qubits, four registers with db qubits, and one register with b qubits, apart from any ancillas used in arithmetic operations, for a total of $2 \log_2 N + (4d + 1)b$ qubits. For $N = 128$ particles (after the doubling trick), which is reasonable for most e^+e^- applications, this is around 300 qubits. Such a device is not far beyond current ≈ 50-qubit computers, so it is naively plausible that the first quantum computer able to run the sequential quantum thrust algorithm (without error correction) could be ready in time to compute realistic thrust distributions at a future e^+e^- collider. Of course, this depends on the gate connectivity of such a device as well as the achievable coherence time, and as discussed below, circuit depth may be more constraining than the number of qubits. For the parallel architecture, we need $N(\log_2 N + db)$ additional qubits for initial data loading [see Eq. (40)], though more qubits would most likely be required to simulate full connectivity and to store intermediate steps of the SUM operation. This points to an $O(10^4)$ qubit device, which is rather optimistic on the 20 year timescale, though this could be made more realistic by preclustering particles to reduce N or by using a smaller value of b.

Next, we consider the number of gates required by the Grover-based thrust algorithm. We first apply $2 \log_2(2N)$ Hadamard gates to obtain the initial state, a uniform superposition over the indices i, j. We then apply $O(N)$ iterations of the Grover operator G, where G consists of two reflections: the reflection over all states with a thrust value greater than the current maximum, an operation requiring the subroutine COMP_T, and the reflection about the initial state. Note that the reflection about the initial state can be effected with an application of $H^{\otimes 2N}$, followed by a reflection about the all-zeros state, followed by an application of $H^{\otimes 2N}$. The Hadamards require $4 \log_2 2N$ gates total, while the reflection about the all-zeros state can be obtained using a controlled-Z operator controlled on having the state $|0\rangle$ in the first $\log_2 N$ registers, which requires $\log_2 N$ CNOT gates. Similarly, after performing COMP_T, we can perform the reflection over all states with a thrust value greater than the current maximum using a controlled-Z operator controlled on the b bits representing the thrust value, an operation requiring b CNOT gates. Thus, the total gate usage of the algorithm scales like
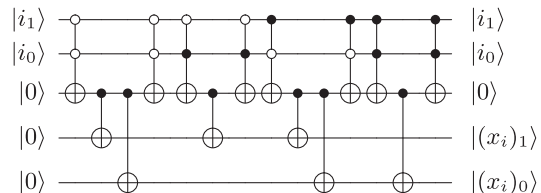

FIG. 5. An example loading circuit mapping $|i\rangle|0\rangle|0\rangle \mapsto |i\rangle|0\rangle|x_i\rangle$. In our example, $i = i_1 i_0$ is two bits and $(x_0, x_1, x_2, x_3) = (3, 2, 3, 1)$. The CCNOT gates are drawn with open (closed) circles if they are controlled on the source bit being zero (one).

$O(N(\log_2 N + C_{\text{COMP\_T}} + b))$, where $C_{\text{COMP\_T}}$ is the gate cost of the COMP_T subroutine.

What is COMP_T? To estimate this, we consider the steps in COMP_T from Fig. 4, noting that these steps consist of either data loading operations like LOOKUP and SUM, or elementary arithmetic operations like addition, multiplication, and division.

In step 1, we load $\vec{p}_i$, $\vec{p}_j$ using LOOKUP. Note that the circuit that implements this looks like the following: first, we have an ancilla bit controlled on each bit in the index register $|i\rangle = |i_{\log_2 N} \ldots i_2 i_1 i_0\rangle$; that is, we have a $C^{\log_2 N}$ NOT gate connecting the ancilla to each index register $|i_k\rangle$. This requires a total of $\log_2 N$ CNOT gates [73]. Then, controlled on whether or not the ancilla bit is set, we want to transform the blank register $|\vec{0}\rangle$ into the register $|\vec{p}_i\rangle$. We set each bit of $\vec{p}_i$ controlled on whether or not the ancilla bit is set, so in total we require db CNOT gates. Finally, we uncompute the ancilla bit by again applying the $C^{\log_2 N}$ NOT gate connecting the ancilla to the $|i\rangle$ register, again requiring $\log_2 N$ gates. In Fig. 5, we give an example circuit for $i = i_1 i_0$, indexing two bits corresponding to items 0, 1, 2, 3 with example values. We have such a circuit for all indices i, requiring $O(N(\log_2 N + db))$ gates in total. For fault-tolerant quantum computers, this procedure can be further optimized [74], but this does not significantly change the resource scaling.

The remaining steps in COMP_T involve performing basic arithmetic operations like addition, multiplication, and division. Circuits for elementary operations like addition and multiplication can be found in Ref. [75], while fault-tolerant versions can also be found in the literature [76,77]. Note that for an input of n bits, addition requires $O(n)$ gates, while multiplication and division require $O(n^2)$ gates.[5] Steps 2 and 4 in COMP_T involve a series of multiplications and divisions with $n = db$ bits, thus requiring $O(d^2 b^2)$ gates. In step 3, we apply SUM controlled on the sign of each $\hat{r}_{ij} \cdot \vec{p}_k$. Here, we first compute each $\hat{r}_{ij} \cdot \vec{p}_k$

---

[5]Asymptotically faster multiplication circuits exist, but they do not yet outperform the $O(n^2)$ algorithm until $n \sim 10^{3-4}$; we thank Craig Gidney for pointing this out.

and then set an ancilla bit depending on the sign of the dot product, requiring $O(Nd^2b^2)$ gates total. Next, for each $\vec{p}_k$, we need to both load the value (using a circuit similar to the one from step 1, requiring $O(N(\log_2 N + db))$ gates total), and then add it to a running sum using an adder circuit if the ancilla bit is set, requiring $O(N(\log_2 N + db))$ gates total. Thus, $C_{\text{COMP\_T}} = O(N(\log_2 N + d^2b^2))$, and the total gate usage of the entire algorithm will scale like $O(N^2(\log_2 N + d^2b^2))$.

Finally, we consider circuit depth, which involves considering which gates can be run in parallel. Note that the $O(N)$ Grover iterations G must come one after the other. Likewise, within each Grover iteration G, the two reflections must come after each other. The parallelization happens within the subroutine COMP_T, where we can parallelize LOAD and SUM in the parallel computing model via preloading; that is, we execute the loading circuits in parallel so that all the $\vec{p}_i$ are in memory, and then we process the $\vec{p}_i$ in parallel.

First, we perform all N preloads in parallel, resulting in a gate depth of $2\log_2 N + db$ gates; this involves performing all N operations in the sequential LOAD operation at once. After everything has been preloaded in parallel memory, we can perform either LOAD or SUM. To perform LOAD, we want to execute a series of $\log_2 N$ swaps and then a copy, which requires $O(\log_2 N + db)$ CNOT gates, so that the whole LOAD operation has a depth of $O(\log_2 N + db)$. Meanwhile, to perform the SUM operation after everything has been preloaded in parallel memory, we note that we must execute the parallel LOAD operation for each $\vec{p}_k$, then calculate and control on the quantity $\hat{r}_{ij} \cdot \vec{p}_k$ for each $\vec{p}_k$, and then we must finally sum all the N vectors. The parallel load requires a gate depth of $O(\log_2 N + db)$, while the dot product calculation requires a gate depth of $O(d^2b^2)$. Finally, we need to perform a series of $\log_2 N$ additions, which requires $db \log_2 N$ gates. Thus, the SUM operation requires a total circuit depth of $O(db \log_2 N + d^2b^2)$. Then $C_{\text{COMP_T}} = O(db\log_2 N + d^2b^2)$, and the circuit depth of the entire parallel algorithm scales like $O(N(db \log_2 N + d^2b^2))$. Note that for the sequential model, the circuit depth is just the same as the gate count of $O(N^2(\log_2 N + d^2b^2))$ since we are not running operations in parallel.

Thus, again taking an example with $N = 128$ particles (after the doubling trick), we would expect a circuit depth of around 10^7 gates for the sequential model and 10^5 for the parallel model. On a noisy device, we currently do not expect to be able to execute an algorithm requiring more than 10^3 gates [78], so again we believe that preclustering particles to reduce N or using a small value of b could make these algorithms more realistic on a NISQ device. We note that because circuit depth and qubit usage come at a tradeoff, circuit depth is the limiting factor for the sequential model, while qubit usage is the limiting factor for the parallel model.

VI. IS THERE A QUANTUM ADVANTAGE?

Starting from the previously best known $O(N^3)$ classical algorithm on a sequential computer, we found an improved $O(N^2 \log N)$ classical algorithm and an $O(N^2)$ quantum algorithm. Because these scalings are identical up to a $\log N$ factor, one might wonder if there is any real quantum advantage for the task of hemisphere jet finding.

Formally, there is a quantum advantage if we make a rather restricted assumption about the computing model. The sequential quantum computing model in Sec. V C only requires read access to the $O(N)$ classical dataset, whereas the sorting strategy in Sec. III C requires write access to $O(N \log N)$ classical bits. Thus, if one restricts the computing model to have write access to only $O(\log N)$ classical bits, then the classical sorting strategy cannot be implemented. In that case, the best classical algorithm would be the $O(N^3)$ one from Ref. [2], which would be bested by our $O(N^2)$ quantum algorithm.

For any realistic application of thrust, this computing model is overly limited, since data from a single collider event can easily be read into random-access classical memory. On the other hand, it is not possible to read in the entire LHC dataset into memory, and indeed some collider datasets are only stored on tape drives. For this reason, there may be interesting quantum advantages for clustering algorithms that act on ensembles of events (instead of on ensembles of particles in a single event). See Ref. [79] for recent developments along these lines.

For the parallel computing models, there is no formal limit with a quantum advantage, since we need $\tilde{O}(N)$ (qu) bits with read-write access in both the quantum and classical cases. Note that the speed up in the classical and quantum cases come from rather different sources. Classical sorting splits the $O(N^2)$ search space into an $O(N)$ outer loop and an $O(\log N)$ inner loop. By contrast, the quantum algorithm searches the $O(N^2)$ search space as a whole in $O(\sqrt{N^2})$ runtime.

This last observation suggests that for even larger search spaces, there might be a quantum advantage even if there exist classical sorting strategies. If classical sorting can only sort s of the search dimensions, then for an $O(N^\alpha)$ search space, the classical runtime would scale proportional to $O(N^{\alpha-s} \log^s N)$. The quantum runtime would scale proportional to $O(N^{\alpha/2})$, which would be faster than the classical case for $\alpha > 2s$. This might be relevant for the M-jet finding problem mentioned in Sec. VII D with an $O(N^{2M})$ search space.

VII. GENERALIZATIONS

In this section, we discuss how to apply the quantum algorithms from Secs. IV and V to jet identification methods that generalize thrust. These algorithms are more closely related to the ones used at the LHC, since they involve a jet radius parameter R.
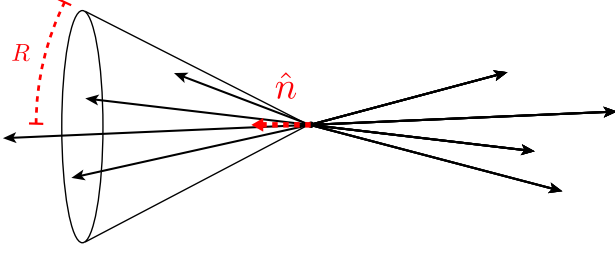
FIG. 6. Partitioning an event into a stable cone jet of radius R and an unclustered region. This is the same as Fig. 1 when $R = \pi/2$.

We start with algorithms that divide the event into a single jet and an unclustered region, as in Fig. 6. (For the thrust problem, $R = \pi/2$ and the unclustered region is the opposite hemisphere.) We then mention strategies to identify multiple jets. To simplify the discussion, we continue to use the (p^x, p^y, p^z) coordinate system for electron-positron collisions, noting that the methods below can be adapted to the standard proton-proton coordinate system of transverse momentum (p_T), rapidity (y), and azimuth (ϕ).

A. SingleCone

The generalizations we consider are all based on or inspired by the analysis of Ref. [62], which showed that the thrust duality in Sec. II D holds for a one-parameter family of jet finding algorithms. No matter which dual formulation is used, we refer to this jet finding strategy as SINGLECONE, since it finds a single stable cone jet of radius R.

To match the literature, we use four-vector notation in this section. The four-momentum of a particle is

$$p_i^\mu = (E_i, \vec{p}_i), \qquad (41)$$

where the energy $E_i \equiv p_i^0 = \sqrt{\vec{p}_i^2 + m_i^2}$ depends on the mass m_i of particle i. The four-momentum of a candidate partition H is

$$P^\mu = \sum_{i \in H} p_i^\mu \equiv (E, \vec{P}), \qquad (42)$$

where $E \equiv P^0$ is the total energy of the partition. A lightlike axis is given by

$$n^\mu = (1, \hat{n}), \qquad (43)$$

with $\hat{n}^2 = 1$. We contract indices with the mostly minus metric,

$$p_\mu q^\mu = p^0 q^0 - \vec{p} \cdot \vec{q}. \qquad (44)$$

The SINGLECONE jet finder is based on maximizing the following objective function [62]:

$$O(P^\mu, n^\mu) = E - \frac{n_\mu P^\mu}{1 - \cos R} + \lambda(n_\mu n^\mu),$$
$$= \frac{\hat{n} \cdot \vec{P} - E \cos R}{1 - \cos R} + \lambda(\hat{n}^2 - 1), \qquad (45)$$

where λ is again a Lagrange multiplier, and we maximize over both the choice of partition and the choice of axis. The second line makes it clear that $R = \pi/2$ returns the thrust objective function in Eq. (17).

Performing the same manipulations as in Sec. II D, the optimum axis (for fixed partition) is

$$n_{\text{opt}}^\mu = \left(1, \frac{\vec{P}}{|\vec{P}|}\right). \qquad (46)$$

Since the optimum axis is aligned with the jet three-momentum, this is an example of a stable cone algorithm; see Sec. VII C below. The reduced SINGLECONE objective function is

$$O(P^\mu) \equiv O(P^\mu, n_{\text{opt}}^\mu) = \frac{|\vec{P}| - E \cos R}{1 - \cos R}, \qquad (47)$$

which is an example of a jet function maximization algorithm [43–45]. The optimum solution partitions the event into a clustered region H and an unclustered region (the complement of H). This definition of the problem naturally lends itself to quantum annealing in Sec. VII B.

Doing the dual manipulation, the optimum partition (for fixed axis) is

$$P_{\text{opt}}^\mu = \sum_{i=1}^{N} p_i^\mu \Theta(E_i(1 - \cos R) - n_\mu p_i^\mu). \qquad (48)$$

Writing the Heaviside theta function requirement in three-momentum language,

$$\frac{\hat{n} \cdot \vec{p}_i}{E_i} > \cos R, \qquad (49)$$

we see that for massless particles ($E_i = |\vec{p}_i|$), the jet constituents are those within an angular distance R of the jet axis. For $R = \pi/2$, this yields the thrust hemisphere regions. The reduced SINGLECONE objective function is now

$$O(n^\mu) \equiv O(P_{\text{opt}}^\mu, n^\mu)$$
$$= \sum_{i=1}^{N} E_i - \sum_{i=1}^{N} \min\left\{E_i, \frac{n \cdot p_i}{1 - \cos R}\right\}, \qquad (50)$$

where we dropped the Lagrange multiplier term for compactness. The second term in Eq. (50) is an example of an N-jettiness measure [80–82] with $N = 1$, whose

minimum yields the XCone jet algorithm [32,33]. This definition of the problem naturally lends itself to quantum search in Sec. VII C.

B. Jet function maximization

In the jet function maximization approach of Refs. [43–45], the goal is to optimize P^μ for a global jet function. The original jet function from Ref. [43] can be written as

$$O_{\text{Georgi}}(P_\mu) = E - \frac{1}{2(1 - \cos R)} \frac{M^2}{E}, \qquad (51)$$

where the jet mass is

$$M^2 \equiv P_\mu P^\mu = E^2 - \vec{P}^2. \qquad (52)$$

In the limit $M \ll E$, this matches the reduced SINGLECONE objective function of Eq. (47), though they yield different optimal jet regions for finite-mass jets.

Since jet function maximization is a kind of partitioning problem, it is natural to try to write these objective functions in QUBO form. However, the original jet function from Eq. (51) is not quadratic since it involves a $1/E$ factor, and the SINGLECONE function in Eq. (47) is not quadratic since $|\vec{P}|$ involves a square root. Thus, these cannot be rewritten as QUBO problems without some kind of modification.

In the analysis of Sec. IV for thrust, we got around this issue by squaring the thrust objective function, which nevertheless yielded the same partitioning solution. This approach does not work in this more general case because of nonquadratic cross terms.

What we can do, however, is square the SINGLECONE objective in Eq. (47) but only keep the lowest nontrivial term in the $M \ll E$ limit.[6] (Squaring and expanding Eq. (51) yields the same result.) This gives the following QUBO objective function:

$$\begin{aligned} O_{\text{QUBO}}(P_\mu) &= E^2 - \frac{M^2}{1 - \cos R} \\ &= \frac{\vec{P}^2 - E^2 \cos R}{1 - \cos R} \\ &= \sum_{i,j=1}^{N} \left(\frac{\vec{p}_i \cdot \vec{p}_j - E_i E_j \cos R}{1 - \cos R} \right) x_i x_j, \quad (53) \end{aligned}$$

where again $x_i \in \{0, 1\}$. Taking $R = \pi/2$ in Eq. (53) then recovers the thrust (squared) problem. It is interesting that Eq. (53) has the same form as the generalized jet functions in Ref. [44] (Ref. [45]) with $n = 2$ ($\alpha = 2$).

This objective function corresponds to a QUBO problem and can thus be solved on a quantum annealer. It will,

---

[6]We thank Eric Metodiev for discussions related to this point.

however, generally yield a different solution compared to SINGLECONE. Unlike SINGLECONE, which yields perfectly conical jets for massless particles via Eq. (49), this QUBO jet finder has an effective jet radius that depends on the mass of the jet [44,45]. Quadratic objective functions are also explored in Ref. [47] for jet clustering at the LHC. In future work, we plan to characterize the general phenomenological properties of jets identified using QUBO objectives.

C. Stable cone finding

Stable cone algorithms search over candidate jet regions of radius R and select ones that are stable [34,83], meaning that the center of the jet region aligns with the jet momentum. As shown in Eqs. (46) and (49), SINGLECONE is an example of a stable cone algorithm, which is closely related to SISCone [3].

It is worth emphasizing two key differences between SINGLECONE and SISCone. First, SINGLECONE finds a single jet, whereas SISCone finds all stable cones, and a separate split/merge step is needed to determine the final jet regions. That said, it is possible to run SISCone in *progressive removal* (PR) mode, where one finds the most energetic stable cone, removes the found jet constituents, and repeats the SISCone procedure on the unclustered particles. In this way, SISCone-PR acts like an iterated application of SINGLECONE. Second, SINGLECONE finds the jet region with the largest value of Eq. (47) ($= E - O(M^2/E)$), whereas SISCone-PR would typically take the stable cone with the largest plain energy E. As we will see below, though, it is still possible to develop quantum algorithms for stable cones with alternative jet hardness sorting schemes.

It is straightforward to implement the SINGLECONE algorithm (a.k.a. SISCone-PR with Eq. (47) ordering) via quantum search. Just as two points define a partitioning plane, two points are enough to determine a cone region of radius R [3]. (This is true up to an eightfold ambiguity, which is twice that of the thrust case because the two candidate cones are not complements of each other as they are for hemispheres.) We can use the LOOKUP operation to determine all $O(N^2)$ candidate reference axes (which are not the same as the jet axes, but yield the same partitions). We can then use SUM to calculate Eq. (47) for a fixed reference axis, since finding P^μ for the particles in the candidate jet region is a linear operation. We finally use Grover search to find the partition that maximizes Eq. (47), and we are guaranteed that the found cone jet will be stable via Eq. (46). This algorithm now has the identical structure to thrust, with the same asymptotic scaling as in Table I, taking us from a classical $O(N^3)$ algorithm (without sort) to a quantum $O(N \log N)$ algorithm (with parallel data loading).

Note that the quantum maximum finding algorithm only returns one maximum element of an array, so we cannot use it to speed up an algorithm for identifying all stable cones.

We can, however, use it to find one stable cone with a different objective function from Eq. (47). For example, to implement SISCone-PR with standard energy ordering, we can use a subroutine consisting of two SUM operations in series. The first SUM determines P^μ for the candidate jet region, while the second SUM finds $\tilde{P}^\mu$ for all particles within a radius R of P^μ. This subroutine would return P^0 if $P^\mu = \tilde{P}^\mu$, while it would return 0 if $P^\mu \neq \tilde{P}^\mu$. One would then use Grover search to find the maximum subroutine output, with the same asymptotic quantum gains as in the SINGLECONE case.

D. Multiregion optimization

Typical collider studies involve more than one jet per event, so it is interesting to ask whether these quantum methods can be adapted to the multijet case. As already mentioned, one can use a PR strategy to identify multiple jet regions, so finding M jets just requires M iterations of the algorithms above. Except in specialized circumstances, the number of desired jet regions does not grow with N and is at most $O(1/R^2)$, so the runtime of SINGLECONE-PR would scale linearly with M. That said, we are interested in simultaneously optimizing the jet regions as in XCone algorithm [32,33], in order to treat the overlapping jet regions in a more sophisticated way than just PR.

The QUBO objective in Eq. (53) can be easily generalized to the M-jet case using $O(N(M + 1))$ qubits, suitable for quantum annealing. Instead of a binary assignment of each particle to the clustered or unclustered region, we can do a one-hot encoding with $M + 1$ qubits per particle to indicate their assignment to one of the M jet regions or to the unclustered region. Specifically, let $x_{ir} \in \{0, 1\}$ for $i \in \{1, ..., N\}$ and $r \in \{0, 1, ..., M\}$. We assign $x_{i0} = 1$ if particle i is in the unclustered region, $x_{ir} = 1$ if particle i is in jet region r for $r \in \{1, ..., M\}$, and $x_{ir} = 0$ otherwise. We then add a penalty term to the objective function such that, for fixed i, $x_{ir} = 1$ for only one value of r.

The multijet QUBO objective function is

$$O_{\text{QUBO}}(\{x_{ij}\}) = \sum_{r=1}^{M} \sum_{i,j=1}^{N} \left(\frac{\vec{p}_i \cdot \vec{p}_j - E_i E_j \cos R}{1 - \cos R} \right) x_{ir} x_{jr}$$
$$+ \Lambda^2 \sum_{i=1}^{N} \left(1 - \sum_{r=0}^{M} x_{ir} \right)^2. \tag{54}$$

Here, there is a copy of Eq. (53) for each of the M jet regions, taking the schematic form of $O = -\sum_{i,j} Q_{ij} x_i x_j$. The coefficient of the penalty term must be taken to be $\Lambda^2 > N \max_{ij} Q_{ij}$ to ensure that it is never favorable for a particle to be assigned to more than one jet region. Because Eq. (54) is quadratic in the momentum, it will not have the same behavior as XCone (which has a linear objective function), though we expect the results to be similar for

well-separated jets of comparable energies. This objective function does not penalize empty jet regions, so it might be interesting to run this algorithm with a large value of M to let the number of nonempty jet regions be determined dynamically.

Compared to the single-jet case, the multijet case will likely be more difficult to implement on currently available quantum annealing hardware. Previous numerical studies [55] have shown that clustering problems that use multiple qubits to implement one-hot encoding are prone to errors. The reason is that on annealing hardware, qubit couplings have a maximum dynamic range, which in turn limits the effectiveness of the Λ penalty term. In practice, this means that annealers often output a fuzzy assignment rather than a hard assignment to one cluster. We would also like to argue that this problem is conceptual in origin. The search space of the single-jet QUBO problem is 2^N, whereas the search space of the multijet QUBO problem is 2^{MN}. However, the QUBO quantum search space contains many extra unphysical states, since the actual (non-QUBO) search space is size $M^N = 2^{N \log M}$. While the most natural way to address this would be to use qudits with $d = M$ instead of qubits, such hardware is not currently available.

Turning to the quantum search case, finding M conical jet regions naively requires searching a space of $O(N^{2M})$, with the added complication of needing to treat overlapping jet regions. We are unaware of any classical approach to this problem apart from brute force, though one expects an $O(N^{2M+1})$ algorithm for the XCone objective should be feasible, though it likely requires a more sophisticated treatment of reference axes. (The current implementation of XCone in FASTJET CONTRIB 1.041 [31,84] only finds a local minimum starting from suitable seed axes.) Using quantum search with sequential (parallel) data loading, one might hope that this could be improved to $O(N^{M+1})$ ($O(N^M \log N)$), though one would have to generalize the LOOKUP and SUM operations to deal with the multijet case. At minimum, LOOKUP would have to load the momenta into $2M$ registers (to label the candidate partitions), and SUM would have to have M distinct outputs (for each of the M jet regions). Even with quantum gains, this is computationally daunting, motivating future studies of multijet algorithm whose computational complexity grows only polynomially with M.

VIII. CONCLUSIONS

In this work, we demonstrated how quantum computers could be applied to a realistic collider physics problem, which requires interfacing a classical dataset with a quantum algorithm. We focused on maximizing thrust to identify hemisphere jets, but the quantum methods developed here are relevant to a broader range of optimization and cluster-finding problems. The asymptotic performance of our quantum annealing and quantum search algorithms

is summarized in Table I. We found a way to improve the previously best known $O(N^3)$ classical thrust algorithm to an $O(N^2)$ sequential quantum algorithm. Along the way, we found an improved $O(N^2 \log N)$ classical algorithm, based on the sorting strategy of Ref. [3]. Both the quantum and improved classical algorithms can be implemented on parallel computing architectures with asymptotic $O(N \log N)$ runtime. Formally, we found a quantum advantage, but only when assuming a computing model with read access to $O(N)$ (qu)bits but write access to only $O(\log N)$ (qu)bits.

Going beyond thrust, we briefly generalized our quantum methods to handle structurally similar jet clustering algorithms. These involve maximizing an objective function with a radius parameter R, which partitions the event into a conical jet region and an unclustered region. While we focused on electron-positron collisions, it is known how to adapt these methods to proton-proton collisions [45,47,62]. In future work, we plan to investigate the phenomenological performance of these "quantum friendly" jet algorithms at the LHC, to assess whether they offer improved physics performance relative to hierarchical clustering schemes like anti-k_t.

The main take home message from this work is that the overhead of data loading must be carefully accounted for when evaluating the potential for quantum speedups on classical datasets. In many ways, optimization-based jet algorithms are an ideal platform to think about quantum algorithms for collider physics, since these problems tend to involve searching over a large space of possibilities, $O(N^\alpha)$ with $\alpha \geq 2$, and therefore benefit from Grover search methods. By contrast, even though the number of events in a collider data sample (N_{events}) is usually much larger than the number of final-state particles in a jet, typical collider tasks like filling a histogram involve $O(N_{events})$ operations, such that data loading is already the limiting factor. On the flip side, this motivates further quantum investigations into classically $O(N_{events}^2)$ data manipulation strategies, such as the metric space approach recently proposed in Ref. [79], since they might be reducible to $O(N_{events})$ quantum algorithms under suitable circumstances. We also note that Grover search is limited to a square-root speedup on unstructured search, whereas collider data have additional structures like symmetries and heuristics which might lead to further quantum gains.

ACKNOWLEDGMENTS

[1] G. P. Salam, Towards jetography, Eur. Phys. J. C **67,** 637 (2010).

[2] H. Yamamoto, An efficient algorithm for calculating thrust in high multiplicity reactions, J. Comput. Phys. **52,** 597 (1983).

[3] G. P. Salam and G. Soyez, A practical seedless infrared-safe cone jet algorithm, J. High Energy Phys. 05 (2007) 086.

[4] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition* (Cambridge University Press, New York, 2011), 10th ed.

[5] V. Giovannetti, S. Lloyd, and L. Maccone, Quantum Random Access Memory, Phys. Rev. Lett. **100,** 160501 (2008).

[6] T. Kadowaki and H. Nishimori, Quantum annealing in the transverse Ising model, Phys. Rev. E **58,** 5355 (1998).

[7] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser, Quantum computation by adiabatic evolution, arXiv:quant-ph/0001106.

[8] S. Boixo, T. Ronnow, S. Isakov, Z. Wang, D. Wecker, D. Lidar, J. Martinis, and M. Troyer, Evidence for quantum annealing with more than one hundred qubits, Nat. Phys. **10,** 218 (2014).

[9] L. K. Grover, A fast quantum mechanical algorithm for database search, in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, STOC '96* (ACM, New York, 1996), pp. 212–219.

[10] M. Boyer, G. Brassard, P. Hoyer, and A. Tapp, Tight bounds on quantum searching, Fortschr. Phys. **46,** 493 (1998).

[11] C. Durr and P. Hoyer, A quantum algorithm for finding the minimum, arXiv:quant-ph/9607014.

[12] S. Brandt, C. Peyrou, R. Sosnowski, and A. Wroblewski, The principal axis of jets. An attempt to analyze high-energy collisions as two-body processes, Phys. Lett. **12,** 57 (1964).

[13] E. Farhi, A QCD Test for Jets, Phys. Rev. Lett. **39,** 1587 (1977).

[14] A. De Rujula, J. R. Ellis, E. G. Floratos, and M. K. Gaillard, QCD predictions for hadronic final states in e^+e^- annihilation, Nucl. Phys. **B138**, 387 (1978).

[15] D. P. Barber *et al.*, Tests of quantum chromodynamics and a direct measurement of the strong coupling constant α_S at $\sqrt{s}$ = 30-GeV, Phys. Lett. **89B**, 139 (1979).

[16] W. Bartel *et al.* (JADE Collaboration), Observation of planar three jet events in e^+e^- annihilation and evidence for gluon bremsstrahlung, Phys. Lett. **91B**, 142 (1980).

[17] M. Althoff *et al.* (TASSO Collaboration), Jet production and fragmentation in e^+e^- annihilation at 12-GeV to 43-GeV, Z. Phys. C **22**, 307 (1984).

[18] D. Bender *et al.*, Study of quark fragmentation at 29-GeV: Global jet parameters and single particle distributions, Phys. Rev. D **31**, 1 (1985).

[19] G. S. Abrams *et al.* (MARK-II Collaboration), First Measurements of Hadronic Decays of the Z Boson, Phys. Rev. Lett. **63**, 1558 (1989).

[20] Y. K. Li *et al.* (AMY Collaboration), Multi-hadron event properties in e^+e^- annihilation at $\sqrt{s}$ = 52 GeV to 57-GeV, Phys. Rev. D **41**, 2675 (1990).

[21] D. Decamp *et al.* (ALEPH Collaboration), Measurement of α_s from the structure of particle clusters produced in hadronic Z decays, Phys. Lett. B **257**, 479 (1991).

[22] W. Braunschweig *et al.* (TASSO Collaboration), Global jet properties at 14-GeV to 44-GeV center-of-mass energy in e^+e^- annihilation, Z. Phys. C **47**, 187 (1990).

[23] K. Abe *et al.* (SLD Collaboration), Measurement of α_s (M_Z^2) from hadronic event observables at the Z^0 resonance, Phys. Rev. D **51**, 962 (1995).

[24] A. Heister *et al.* (ALEPH Collaboration), Studies of QCD at Z^0 centre-of-mass energies between 91-GeV and 209-GeV, Eur. Phys. J. C **35**, 457 (2004).

[25] J. Abdallah *et al.* (DELPHI Collaboration), A study of the energy evolution of event shape distributions and their means with the DELPHI detector at LEP, Eur. Phys. J. C **29**, 285 (2003).

[26] P. Achard *et al.* (L3 Collaboration), Studies of hadronic event structure in e^+e^- annihilation from 30-GeV to 209-GeV with the L3 detector, Phys. Rep. **399**, 71 (2004).

[27] G. Abbiendi *et al.* (OPAL Collaboration), Measurement of event shape distributions and moments in $e + e \rightarrow$ hadrons at 91-GeV—209-GeV and a determination of α_s, Eur. Phys. J. C **40**, 287 (2005).

[28] M. Cacciari, G. P. Salam, and G. Soyez, The catchment area of jets, J. High Energy Phys. 04 (2008) 005.

[29] M. Cacciari, G. P. Salam, and G. Soyez, The anti-k_t jet clustering algorithm, J. High Energy Phys. 04 (2008) 063.

[30] M. Cacciari and G. P. Salam, Dispelling the N^3 myth for the k_t jet-finder, Phys. Lett. B **641**, 57 (2006).

[31] M. Cacciari, G. P. Salam, and G. Soyez, FastJet user manual, Eur. Phys. J. C **72**, 1896 (2012).

[32] I. W. Stewart, F. J. Tackmann, J. Thaler, C. K. Vermilion, and T. F. Wilkason, XCone: N-jettiness as an exclusive cone jet algorithm, J. High Energy Phys. 11 (2015) 072.

[33] J. Thaler and T. F. Wilkason, Resolving boosted jets with XCone, J. High Energy Phys. 12 (2015) 051.

[34] S. D. Ellis, J. Huston, and M. Tonnesmann, On building better cone jet algorithms, eConf **C010630**, 513 (2001).

[35] C. F. Berger *et al.*, Snowmass 2001: Jet energy flow project, eConf **C010630**, P512 (2001).

[36] L. Angelini, P. De Felice, M. Maggi, G. Nardulli, L. Nitti, M. Pellicoro, and S. Stramaglia, Jet analysis by deterministic annealing, Phys. Lett. B **545**, 315 (2002).

[37] L. Angelini, G. Nardulli, L. Nitti, M. Pellicoro, D. Perrino, and S. Stramaglia, Deterministic annealing as a jet clustering algorithm in hadronic collisions, Phys. Lett. B **601**, 56 (2004).

[38] D. Y. Grigoriev, E. Jankowski, and F. V. Tkachov, Towards a Standard Jet Definition, Phys. Rev. Lett. **91**, 061801 (2003).

[39] D. Y. Grigoriev, E. Jankowski, and F. V. Tkachov, Optimal jet finder, Comput. Phys. Commun. **155**, 42 (2003).

[40] S. Chekanov, A new jet algorithm based on the k-means clustering for the reconstruction of heavy states from jets, Eur. Phys. J. C **47**, 611 (2006).

[41] Y.-S. Lai and B. A. Cole, Jet reconstruction in hadronic collisions by Gaussian filtering, arXiv:0806.1499.

[42] I. Volobouev, FFTJet: A package for multiresolution particle jet reconstruction in the Fourier domain, arXiv:0907.0270.

[43] H. Georgi, A simple alternative to jet-clustering algorithms, arXiv:1408.1161.

[44] S.-F. Ge, The Georgi algorithms of jet clustering, J. High Energy Phys. 05 (2015) 066.

[45] Y. Bai, Z. Han, and R. Lu, J_{E_T}: A global jet finding algorithm, J. High Energy Phys. 03 (2015) 102.

[46] L. Mackey, B. Nachman, A. Schwartzman, and C. Stansbury, Fuzzy jets, J. High Energy Phys. 06 (2016) 010.

[47] Y. Bai, Z. Han, and R. Lu, $J_{E_T}^{II}$: A two-prong jet finding algorithm, arXiv:1509.07522.

[48] A. Mott, J. Job, J. R. Vlimant, D. Lidar, and M. Spiropulu, Solving a Higgs optimization problem with quantum annealing for machine learning, Nature (London) **550**, 375 (2017).

[49] A. Zlokapa, A. Mott, J. Job, J.-R. Vlimant, D. Lidar, and M. Spiropulu, Quantum adiabatic machine learning with zooming, arXiv:1908.04480.

[50] D. Provasoli, B. Nachman, W. A. de Jong, and C. W. Bauer, A quantum algorithm to efficiently sample from interfering binary trees, arXiv:1901.08148.

[51] C. W. Bauer, B. Nachman, D. Provasoli, and W. A. De Jong, A quantum algorithm for high energy physics simulations, arXiv:1904.03196.

[52] I. Shapoval and P. Calafiura, Quantum associative memory in HEP track pattern recognition, EPJ Web Conf. **214**, 01012 (2019).

[53] F. Bapst, W. Bhimji, P. Calafiura, H. Gray, W. Lavrijsen, and L. Linder, A pattern recognition algorithm for quantum annealers, arXiv:1902.08324.

[54] A. Zlokapa, A. Anand, J.-R. Vlimant, J. M. Duarte, J. Job, D. Lidar, and M. Spiropulu, Charged particle tracking with quantum annealing-inspired optimization, arXiv:1908.04475.

[55] V. Kumar, G. Bass, C. Tomlin, and J. Dulny, Quantum annealing for combinatorial clustering, Quantum Inf. Process. **17**, 39 (2018).

[56] F. Neukart, D. Von Dollen, and C. Seidel, Quantum-assisted cluster analysis, arXiv:1803.02886.

[57] S. Brandt and H. D. Dahmen, Axes and scalar measures of two-jet and three-jet events, Z. Phys. C **1**, 61 (1979).

[58] V. Bhattiprolu, M. Ghosh, V. Guruswami, E. Lee, and M. Tulsiani, Inapproximability of matrix $p(\rightarrow)q$ norms, arXiv:1802.07425.

[59] D. Steinberg, Computation of matrix norms with applications to robust optimization, Master's thesis, Technion, 2005.

[60] A. Bhaskara and A. Vijayaraghavan, Approximating matrix p-norms, arXiv:1001.2613.

[61] J. M. Hendrickx and A. Olshevsky, Matrix P-norms are NP-hard to approximate if $p \neq 1, 2, \infty$, arXiv:0908.1397.

[62] J. Thaler, Jet maximization, axis minimization, and stable cone finding, Phys. Rev. D **92**, 074001 (2015).

[63] T. Sjstrand, S. Ask, J. R. Christiansen, R. Corke, N. Desai, P. Ilten, .S. Mrenna, S. Prestel, C. O. Rasmussen, and P. Z. Skands, An introduction to PYTHIA 8.2, Comput. Phys. Commun. **191**, 159 (2015).

[64] T. Sjostrand, S. Mrenna, and P. Z. Skands, PYTHIA 6.4 physics and manual, J. High Energy Phys. 05 (2006) 026.

[65] U. Vishkin, Thinking in parallel: Some basic data-parallel algorithms and techniques (2010), online class notes.

[66] D. M. W. Powers, Parallelized quicksort with optimal speedup, in *Proceedings of the International Conference on Parallel Computing Technologies* (World Scientific, Novosibirsk, 1991).

[67] R. E. Ladner and M. J. Fischer, Parallel prefix computation, J. ACM **27**, 831 (1980).

[68] P. T. Komiske, E. M. Metodiev, and J. Thaler, Cutting multiparticle correlators down to size, to be published.

[69] P. Hoyer, J. Neerbek, and Y. Shi, Quantum complexities of ordered searching, sorting, and element distinctness, Algorithmica **34**, 429 (2002).

[70] A. Montanaro, Quantum walk speedup of backtracking algorithms, arXiv:1509.02374.

[71] A. Montanaro, Quantum speedup of branch-and-bound algorithms, Phys. Rev. Research **2**, 013056 (2020).

[72] S. Brierley, Efficient implementation of quantum circuits with limited qubit interactions, arXiv:1507.04263.

[73] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, Elementary gates for quantum computation, Phys. Rev. A **52**, 3457 (1995).

[74] G. H. Low, V. Kliuchnikov, and L. Schaeffer, Trading T-gates for dirty qubits in state preparation and unitary synthesis, arXiv:1812.00954.

[75] V. Vedral, A. Barenco, and A. Ekert, Quantum networks for elementary arithmetic operations, Phys. Rev. A **54**, 147 (1996).

[76] C. Gidney, Halving the cost of quantum addition, Quantum **2**, 74 (2018).

[77] H. Thapliyal, E. Muoz-Coreas, T. S. S. Varun, and T. S. Humble, Quantum circuit designs of integer division optimizing T-count and T-depth, arXiv:1809.09732.

[78] J. Preskill, Quantum computing in the NISQ era and beyond, Quantum **2**, 79 (2018).

[79] P. T. Komiske, E. M. Metodiev, and J. Thaler, The Metric Space of Collider Events, Phys. Rev. Lett. **123**, 041801 (2019).

[80] I. W. Stewart, F. J. Tackmann, and W. J. Waalewijn, N-Jettiness: An Inclusive Event Shape to Veto Jets, Phys. Rev. Lett. **105**, 092002 (2010).

[81] J. Thaler and K. Van Tilburg, Identifying boosted objects with n-subjettiness, J. High Energy Phys. 03 (2011) 015.

[82] J. Thaler and K. Van Tilburg, Maximizing boosted top identification by minimizing n-subjettiness, J. High Energy Phys. 02 (2012) 093.

[83] G. C. Blazey *et al.*, Run II jet physics, in *Proceedings of the QCD and Weak Boson Physics in Run II. Batavia, USA, 1999* (Fermilab, Batavia, 2000), pp. 47–77.

[84] FastJet Contrib, http://fastjet.hepforge.org/contrib/.