Enumeration on GPU for fplll

Marc Stevens, Simon Pohmann, Jens Zumbrägel

December 20, 2020

1 Approach

1.1 Lattice Enumeration

Given an n-dimensional lattice L with basis $b_1, ..., b_n$, consider the projections π_k into the space $\langle b_1, ..., b_k \rangle^\perp$. Then the idea of lattice enumeration is to begin with the origin $0 \in \pi_n L = \{0\}$ and repeatedly expand points of norm $\leq r$ in $\pi_{k+1} L$ to multiple points of norm $\leq r$ in $\pi_k L$.

For $p \in \pi_{k+1} L$ of norm $\|p\| \leq r$ we call all points $p' \in \pi_k L$ of norm $\|p'\| \leq r$ with $\pi_{k+1} p' = p$ the children of p. As the projections do not increase the norm of vectors, have that for $p \in \pi_k$ with norm $\|p\| \leq r$, also $\pi_{k+1} p \in \pi_{k+1} L$ is of norm $\leq r$. Therefore, considering the children of each point in $\pi_{k+1} L$ of norm $\leq r$ yield exactly all points in $\pi_k L$ of norm $\leq r$.

As the use of "children" already indicates, this defines a tree, the enumeration tree, with root 0 in which each maximal path has length $n+1$, where the leaf nodes are exactly all lattice points of norm $\leq r$. This tree is then searched with a depth-first strategy, and the shortest nonzero leaf node is returned.

Therefore, the fundamental operation of the algorithm is the calculation of all children points. Usually, instead of storing the points $p \in \pi_k L$, the coefficients w.r.t $\pi_k b_i$ are stored. In this case, on tree level $n - k$, each point $p \in \pi_k L$ has a representation $p = \pi_k \sum_{i=k+1}^{n} x_i b_i,\ x_i \in \mathbb{Z}$, so the coefficients for $b_1, ..., b_k$ are zero. It follows that the children p' of a point $p = \pi_k \sum_{i=k+1}^{n} x_i b_i$ are characterized by their coefficient x_k, so they are of the form

$$p' = \pi_{k-1} \sum_{i=k}^{n} x_i b_i \quad \text{where} \quad \left| \|b_k^*\| x_k + \sum_{i=k+1}^{n} x_i \mu_{ki} \right| \leq \sqrt{r^2 - \|p\|^2},\ x_k \in \mathbb{Z}$$

which can be easily seen by calculating $\|p'\|$

$$\begin{aligned}\|p'\| =&\|\pi_{k-1}\sum_{i=k}^{n} x_i b_i\|^2 = \left(\frac{1}{\langle b_k^*, b_k^*\rangle}\langle b_k^*, \sum_{i=k}^{n} x_i b_i\rangle\right)^2 + \|\pi_k \sum_{i=k}^{n} x_i b_i\|^2 \\ =&\left(\sum_{i=k}^{n} x_i \mu_{ki}\right)^2 + \|\pi_k \sum_{i=k+1}^{n} x_i b_i\|^2 = \|p\|^2 + \left(x_k\|b_k^*\| + \sum_{i=k+1}^{n} x_i \mu_{ki}\right)^2\end{aligned}$$

Iterating over all children is therefore equivalent to iterate over all integers x_k between

$$-\sum_{i=k+1}^{n} x_i\mu_{ki} - \sqrt{r^2 - \|p\|^2} \;\le\; x_k \;\le\; -\sum_{i=k+1}^{n} x_i\mu_{ki} + \sqrt{r^2 - \|p\|^2}$$

1.1.1 The partial center sums

The real work is therefore calculating this sum $\sum_{i=k+1}^{n} c_i\mu_{ki}$, often called `center` as it is the center of the interval from which to choose c. By keeping track of all the partial sums $\sum_{i=k}^{n} c_i\mu_{li}$ for $l < k$, the center is always available, and updating the partial sums requires $n - k$ multiplications on tree level $n - k$.

1.1.2 Decrease enumeration bound

When finding any leaf node in the enumeration tree, this node corresponds to a lattice point $x \in L$. If $x \neq 0$, we know that there is a lattice point of norm $\le \|x\|$ in the lattice, so to find the shortest one, it suffices now to search only the points of norm $\le \|x\|$. In other word, we can potentially decrease the enumeration bound r by $r := \min\{r, \|x\|\}$ (there is some complication because of rounding errors during floating point arithmetic, see ToDo), which can significantly reduce the size of the tree. Therefore, finding leaf nodes as early as possible is important for a fast algorithm.

The basic algorithm therefore looks like this

Algorithm 1 Find tree node children

Input: parent coefficients $x_{k+1}, ..., x_n$, parent norm $\|p\|^2$, partial center sums $\sum_i x_i\mu_{li}$ for $l < k$, matrix (μ_{ij})

Set center $:= \sum_{i=k+1}^{n} c_i\mu_{ij}$
Set $x_0 = \lfloor \text{center} \rceil$
Set $\delta = 1$ if center $\ge x_0$, otherwise $\delta = -1$
for all $x \in \{x_0, x_0 + \delta, x_0 - \delta, x_0 + 2\delta, ...\}$ **do**
  If $|x - x_0|^2 > r^2 - \|p\|^2$, exit
  Otherwise, yield the point p' with coefficients $(x, x_{k+1}, ..., x_n)$ and norm $\|p'\|^2 = \|p\|^2 + (x - \text{center})^2\|b_k^*\|^2$
end for

1.2 The CUDA programming model

ToDo

1.3 Parallelization of the Lattice Enumeration

The main difficulty of implementing the enumeration algorithm efficiently on GPUs is the fact that nodes in the enumeration tree have greatly varying degree, so subtrees may have completely different size and structure. This introduces a lot of branching and makes it hard to evenly distribute work on the threads.

These problems especially occur in the following, "naive" approach: Enumerate all points on a certain tree level on the host, and then assign each GPU thread one of these points and let them enumerate the corresponding subtree. Nevertheless, this is still the main idea of our approach. However, we try to counter the problems by assigning a subtree not to a thread, but to a warp and using a work-stealing queue to distribute work among warps.

1.4 Subtree enumeration within a warp

The main idea for the thread cooperation within a warp is to let every thread expand the children of an assigned node, but not recurse into the corresponding subtrees. Instead, all the created new children nodes are then written to memory and are assigned to potentially different threads in the next step.

This prevents threads whose subtrees have different size to diverge and having to wait for the longer one. Additionally, having a list of nodes whose subtrees must still be searched also allows us to pick nodes that will be processed in the next step. This way, we ensure that all threads always work on nodes on the same tree level, which allows coalesced memory access, given a correct memory layout of the data.

The caveat of this approach is of course that it requires frequent memory accesses to load/store the tree nodes. The latency introduced by this is the main factor limiting performance. To at least reduce it, we apply the node shuffling not at each tree level, but only at every k-th tree level, for a constant k (in experiments, $k = 3$ has yielded the best results). In some more detail, this is described in 2. For finding the coefficients of the points $\text{children}^k(\{x_i\})$ in the algorithm, an adaption of the efficient (but branching) recursive enumeration procedure from fplll is used. It also uses the previously calculated partial center sum values (see 1.1.1).

Algorithm 2 Intra-warp enumeration
Input: subtree root r

Init buffer with single node r
while node buffer is not empty **do**
  $l :=$ deepest tree level for which there are ≥ 32 nodes
  If such a l does not exist, use highest level with $\neq 0$ nodes
  Assign one node x_i on level l to each thread $i \in \{0, ..., 31\}$
  if l is leaf level of enumeration tree **then**
    If x_i is nonzero and shorter than the current optimal solution, update it
  else
    Thread i gets all transitive children points children$^k(\{x_i\})$ using Alg. 1
    Store their coefficients and norms
    Calculate new partial center sums with a parallelized matrix-matrix multiplication
  end if
end while

2 Performance

The following benchmark was done on a machine with an Intel core i7-7700K CPU and a GeForce GTX 1080 Ti GPU. As a comparison for the CUDA implementation, we use the multithreaded enumeration algorithm from the fplll library. For each dimension, four knapsack matrices with a uniform 350-bit column were used as lattice, and the graph shows the median of the running time of both implementations. The matrices can also be reproduced using the tool latticegen from the fplll library (via `latticegen -randseed $s r $dim 350` for $s \in \{0, 1, 2, 3\}$). The results are displayed in figure 1.

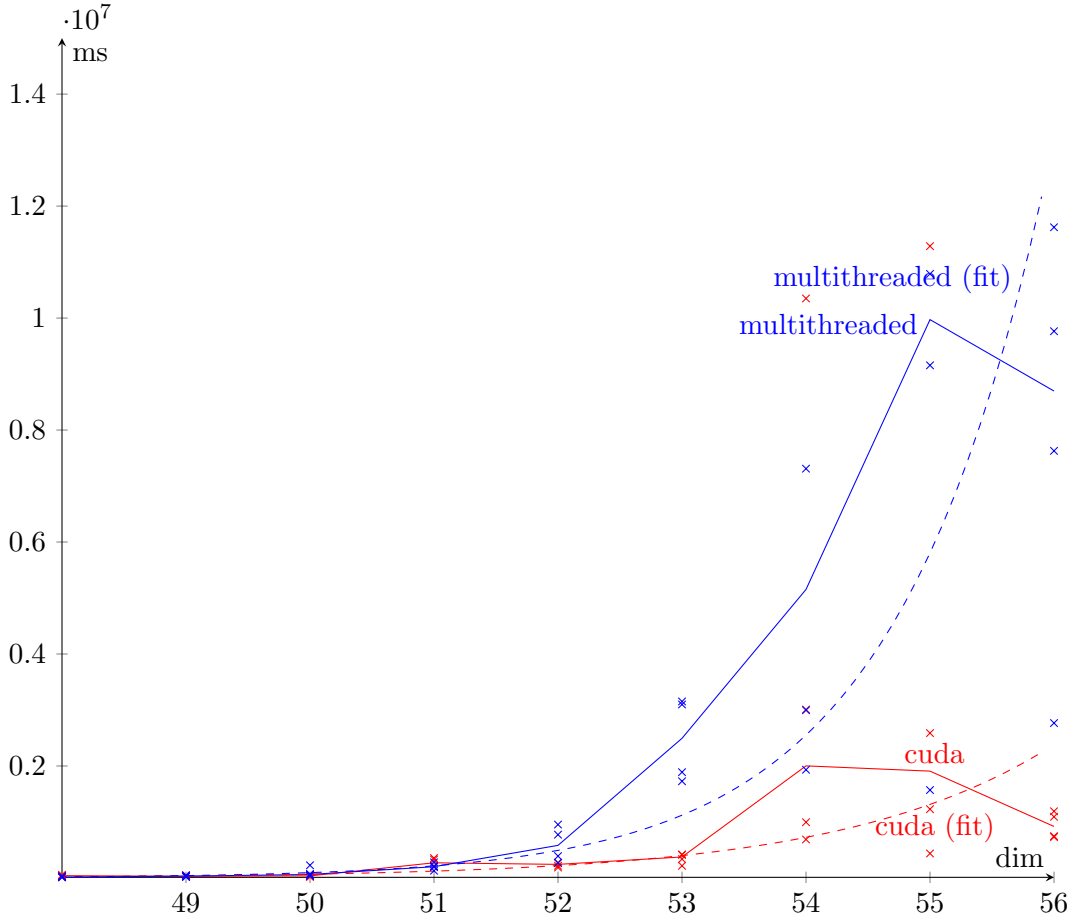

Figure 1: Performance of cuda enumeration (red) and multithreaded enumeration (blue); some data points are clipped