DIMACS Workshop on Parallelism: A 2020 Vision
Lattice Basis Reduction and Multi-Core

Werner Backes and Susanne Wetzel

Stevens Institute of Technology

29th March 2011

Work supported through NSF Grant DUE 1027452.

Introduction

Definition

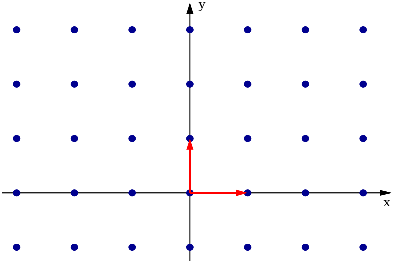
Let $n, k \in \mathbb{N}$ with $k \leq n$. A **lattice** $L \subset \mathbb{R}^n$ is a discrete, additive subgroup of $\mathbb{R}^n$, such that

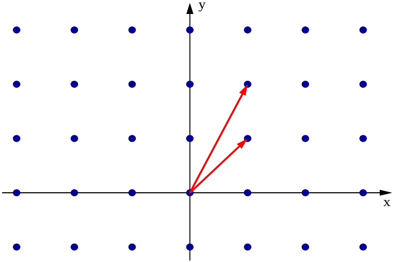
$$L = \{\sum_{i=1}^{k} x_i \underline{b}_i \mid x_i \in \mathbb{Z}, i = 1, \ldots, k\},$$

where $\underline{b}_1, \underline{b}_2, \ldots, \underline{b}_k \in \mathbb{R}^n$ are linearly independent vectors.
$B = (\underline{b}_1, \ldots, \underline{b}_k) \in \mathbb{R}^{n \times k}$ is a **basis** of the k-dimensional lattice L.

Lattice basis reduction: Find a basis $B' = (\underline{b}'_1, \ldots, \underline{b}'_k)$ for lattice $L(B)$ with $BU = B'$ (U unimodular) and as short and orthogonal basis vectors as possible.

Introduction – Example





$$L_1 = \left\{ \begin{pmatrix} x \\ y \end{pmatrix} \mid \begin{pmatrix} x \\ y \end{pmatrix} = c \begin{pmatrix} 1 \\ 1 \end{pmatrix} + d \begin{pmatrix} 1 \\ 2 \end{pmatrix} ; c, d \in \mathbb{Z} \right\}$$

$$L_2 = \left\{ \begin{pmatrix} x \\ y \end{pmatrix} \mid \begin{pmatrix} x \\ y \end{pmatrix} = c \begin{pmatrix} 1 \\ 0 \end{pmatrix} + d \begin{pmatrix} 0 \\ 1 \end{pmatrix} ; c, d \in \mathbb{Z} \right\}$$

Lattice Basis Reduction – Schnorr-Euchner LLL algorithm

Features of Schnorr-Euchner Algorithm

- Overcomes stability/performance issues of the original LLL.
- Uses two representations of the lattice basis.
    - Exact representation (long integer arithmetic) used to perform vector operations.
    - Approximate representation (multi-precision floating point, double) used for orthogonalization and computation of Gram-Schmidt coefficients μ_{ij}.
- Correction steps (heuristics) improve stability of the algorithm.
    - Computation of exact scalar product if necessary.
    - Step back, if size-reduction factors are too big.

Lattice Basis Reduction – Schnorr-Euchner LLL algorithm

Algorithm (Schnorr-Euchner LLL algorithm – simplified)

INPUT: *Lattice basis* $B = (\underline{b}_1, \ldots, \underline{b}_k) \in \mathbb{Z}^{n \times k}$

OUTPUT: *LLL-reduced lattice basis B*

```
APPROX_BASIS(B', B)
while (i ≤ k) do
   SCALAR-PRODUCTS(R, μ, B', B)
   (correction step might require computation of exact scalar product)
   ORTHOGONALIZATION(R, μ)
   μ-UPDATE(μ)
   SIZE-REDUCTION(B, μ)
   (might trigger step back correction step)
   if (F_c = false ∧ F_r = true) then                 // recompute orthogonalization
      ORTHOGONALIZATION(R, μ)
   F_r = false
   if (F_c = true) then                               // do we have to do a step back?
      i = max(i − 1, 2)
      F_c = false
   else
      i' = i                                          // check for LLL condition
      while ((i > 1) ∧ (yR_{i−1,i−1} > S_{i−1})) do
         SWAP(b_i, b_{i−1})
         SWAP(b'_i, b'_{i−1})
         i = i − 1
      if (i ≠ i') then
         if (i = 1) then
            R_11 = ||b'_1||
            i = 2
      else
         i = i + 1
```

Parallel Lattice Basis Reduction – Challenges

Challenges: Dependencies

- Partial, on demand computation of orthogonal basis.
- Might compute exact scalar product as correction step.
- Size-reduction relies on computation of orthogonal basis.
- Single size-reduction updates of orthogonal basis.
- Orthogonal basis is dependent on the order of the basis vector.

Solutions to date:

1. Identify parallel and non-parallel parts within the algorithm.
   - Find alternative ways to perform computations in parallel.
2. Minimize non-parallel portion of the code.
   - Non-parallel portion of the code limits speed-up factor.
3. Balance workload for each parallel part among all threads.
   - Minimize the waiting time at barriers.
   - Minimize the number of barriers and locks.
   - Main thread does prepare for parallel computations.
   - A slight imbalance sometimes helps to keep the balance.

Parallel Lattice Basis Reduction – Scalar Products

Scalar product:

- Computation of scalar product including correction step can be taken out of the orthogonalization and precomputed.
- Divide computation into slices to minimize overhead and compensate for unpredictable correction steps.
- Size of a slice depends on the value of i.

Algorithm (Scalarproducts with correction step)

$s_{sp} = \left(\lceil \frac{i}{n} \rceil > sp_{max}\right) \; ? \; sp_{max} \; : \; \lceil \frac{i}{n} \rceil$

$\mathtt{s} = s_{sp} \cdot (t-1)$, $\mathtt{e} = s_{sp} \cdot t$

while $(\mathtt{s} \leq i)$ **do**

  $\mathtt{e} = (\mathtt{e} > i) \; ? \; i \; : \; \mathtt{e}$

  for $(\mathtt{s} \leq \mathtt{j} < \mathtt{e})$ **do**

    if $(|\langle \underline{b}'_i, \underline{b}'_j \rangle| < 2^{-\frac{p}{2}} \|\underline{b}'_i\| \|\underline{b}'_j\|)$ **then**

      $R_{ij} = \mathrm{APPROX\_VALUE}(\langle \underline{b}_i, \underline{b}_j \rangle)$

    else

      $R_{ij} = \langle \underline{b}'_i, \underline{b}'_j \rangle$

  MUTEX_LOCK(l_1)

  $\mathtt{s} = sl$

  $sl = sl + s_{sp}$

  $\mathtt{e} = sl$

  MUTEX_UNLOCK(l_1)

Parallel Lattice Basis Reduction – Orthogonalization (I)

Orthogonalization:

- Remainder of orthogonalization too hard to parallelize in current form.
- Need to transform computation in order to allow for a parallel computation.

Algorithm (orthogonalization)

Standard implementation:

```
for (1 ≤ j < i) do
    for (1 ≤ m < j) do
        R_ij = R_ij − R_im μ_jm
    μ_ij = R_ij / R_jj
    R_ii = R_ii − R_ij μ_ij
    S_{j+1} = R_ii
```

Parallel enabling version:

```
for (1 ≤ j < i) do
    for (j ≤ l < i) do
        r_l = r_l + R_{i,j−1} μ_{l,j−1}
    R_ij = R_ij − r_j
    μ_ij = R_ij / R_jj
    R_ii = R_ii − R_ij μ_ij
    S_{j+1} = R_ii
```

- The values for r_l can now be computed in parallel.

Parallel Lattice Basis Reduction – Orthogonalization (II)

- Main thread (thread_1) performs precomputation.
- Compute threads wait for the start of the parallel computation.

Algorithm (Orthogonolization – main and compute threads)

Orthogonalization – Thread_1

```
t_a = COMPUTE_ACTIVE_THREAD(i)
j = 0
while (j < i) do
  s = j,  m = 0
  while (m < s_o ∧ j < i) do
    for (s ≤ l < j) do
      r_j = r_j − R_il μ_jl
    R_ij = R_ij − r_j
    μ_ij = R_ij / R_jj
    R_ii = R_ii − R_ij μ_ij
    S_{j+1} = R_ii
    r_j = 0
    m = m + 1,  j = j + 1
  COMPUTE_SPLIT_VALUES_1(split, t_a)
  BARRIER_WAIT(b_1)
  for (j ≤ l < split_1) do
    for (s ≤ m < j) do
      r_l = r_l − R_im μ_lm
```

Orthogonalization – Thread_t

```
t_a = COMPUTE_ACTIVE_THREAD(i)
if (t_a ≤ t) then
  e = 0
  while (e < i) do
    s = e,  e = e + s_o
    if (e > i) then
      e = i
    BARRIER_WAIT(b_1)
    for (split_t ≤ l < split_{t+1}) do
      for (s ≤ m < e) do
        r_l = r_l − R_im μ_lm
```

Parallel Lattice Basis Reduction – Size-Reduction

Size-reduction:

- Vector operations on the exact basis can be parallelized easily.
- Separation into main and compute threads is not necessary.
- Divide vectors into equal sized part.

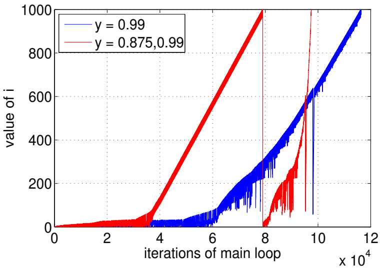
μ-**Update:**

- Similar parallelization method as the one used for the orthogonalization.
- Main thread performs per-computations in order to allow for parallel computations afterwards.
- Dynamically decide on number of active threads depending on value of i.

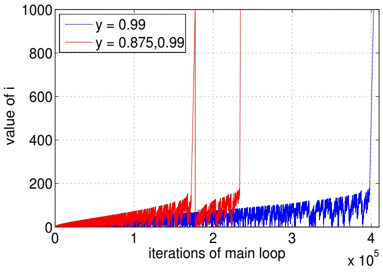
Parallel Lattice Basis Reduction – Sequences of y

Reduction Parameter y:

- Using sequences instead of single parameter y.
- Higher values for i translate into more work that can be parallelized within main while-loop body.
- Minimal effect on single core performance for knapsack and SVP challenge type lattice bases, but beneficial for multi-core.
- More effective for SVP challenge than for knapsack type lattice bases.



knapsack type lattice basis



SVP challenge type lattice basis

Parallel Lattice Basis Reduction – Results (I)

Lattice bases:

- NTRU type lattice bases (cyclic sub-structure), with dimensions 50 to 2000, cyclic sub-structure of size 25 to 1000.
- Knapsack type lattice bases, with entries up to 1000 bit and dimensions from 50 to 2000.
- SVP challenge type lattice bases, dimensions from 50 to 2000.
- Generated and reduced 20 lattice bases for each combination of type, dimension an bit length.

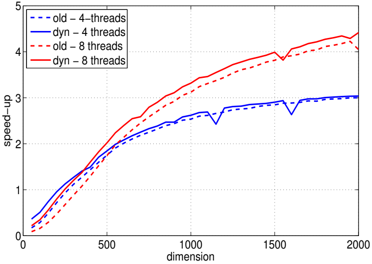
Test systems:

- Sun X4150 Server, 2 x Quad-Core Intel Xeon 2.83 Ghz, 8 GB Ram, Debian Linux.
- These processors do not have a fast interconnect or an integrated memory controller.
- Tested the 4 and 8-thread version of the algorithm.
- Measured real time, user time, and system time.

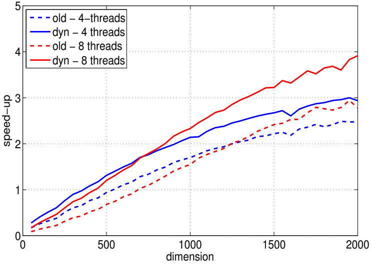
Parallel Lattice Basis Reduction – Results (II)

Speed-up:

- Quotients of user time for the non-parallel and the real time for the multi-threaded version.
- Non-parallel and parallel algorithms testes using sequence of $y = 0.875, 0.99$.



knapsack type lattice basis

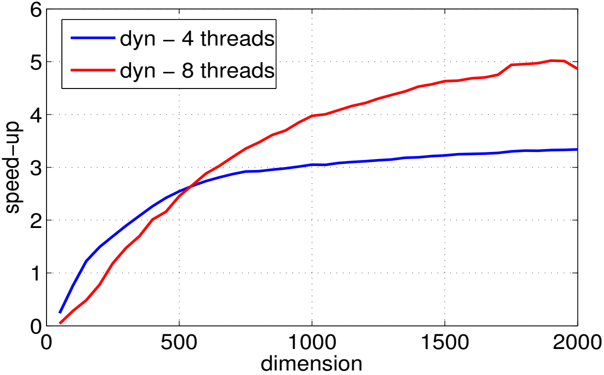


SVP challenge type lattice basis

Parallel Lattice Basis Reduction – Results (III)

Speed-up:

- Knapsack type lattice basis with smaller entries up to 500 bits.
- Doubles instead of multi-precision floating point values used for approximation.
- Two quad-core Intel Xeon CPUs of the newest generation.



Parallel Lattice Basis Reduction – BoostReduce Framework

BoostReduce Framework:

- Alternative method for computing a "good" basis necessary for finding "short" lattice vectors.

Goal:

- Solve the underlying problems with a focus on methods/algorithms that can be parallelized easily.

Approach:

- Uses a new parallel method for finding short lattice vectors.
- Short vectors are integrated into an "improved" lattice basis.
- No overall optimal strategy for the integration of vectors $\Rightarrow$ several "improved" lattice bases are generated.
- Improved lattice bases are then reduced in parallel.
- Best basis out of these reductions is used as starting point for the next rounds in the BoostReduce framework.

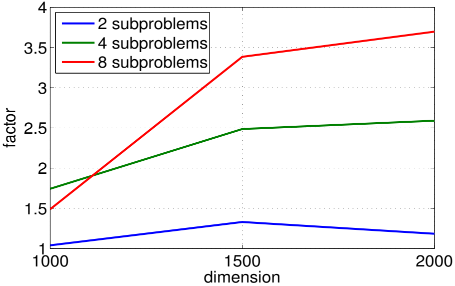
$\Rightarrow$ Find alternative methods to improve the parallel LLL reduction beyond the currently number of 8 or 12 threads.

Parallel Lattice Basis Reduction – Beyond 8 Threads

Beyond 8 **threads:**

- Find ways around the dependencies issues of the LLL.
- Goal is to create independent subproblems that can be solved in parallel without the need for a tight synchronization.
- The independent sub problems should be of equal size in order for them to require a similar reduction times.
- Combine results into LLL reduced basis of the initial lattice.

$\Rightarrow$ Sum of running times has to give us an advantage.



SVP challenge type lattice bases

Parallel Lattice Basis Reduction – Conclusion/Future Work

Conclusion:

- Significantly improved our parallel LLL based on the Schnorr-Euchner algorithm.
- Sequences of reduction parameters beneficial in parallel case.
- Dynamically adjust parameters used to divide the work load depending on value of i (main loop).
- BoostReduce is a first alternative approach to lattice basis reduction.
- Encouraging initial results for parallel LLL reduction of certain lattice basis types beyond 8 threads.

Future work:

- Further improve the parallel LLL for lattice bases with lower dimension and smaller entries (use a different approximation).
- New scheduling for orthogonalization and μ-update.
- Improve on the alternative reduction algorithms.
- Parallelize stronger reduction algorithms, such as BKZ.

Parallel Lattice Basis Reduction – References

References:

1. *Factoring Polynomials with Rational Coefficients*, A.K. Lensta, H.W. Lenstra, and L. Lovász, Math. Ann. Volume 261, 1982.
2. *Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems*, C.P. Schnorr and M. Euchner, Proceedings of FCT 91, 1991.
3. *Floating-Point LLL Revisited*, P. Nguyen and D. Stéhle, Proceedings of Eurocrypt 2005.
4. *BoostReduce - A Framework For Strong Lattice Basis Reduction*, Werner Backes and Susanne Wetzel, ePrint 2010/386
5. *Parallel Lattice Basis Reduction using a Multi-Threaded Schnorr-Euchner LLL Algorithm*, Werner Backes and Susanne Wetzel, Proceedings of Euro-Par 2009, Delft.
6. *Heuristics on Lattice Basis Reduction in Practice*, Werner Backes and Susanne Wetzel, ACM Journal on Experimental Algorithms, Vol. 7, 2002.