Shortest Lattice Vector Enumeration on Graphics Cards

Jens Hermans [1] Michael Schneider [2] Fréderik Vercauteren [1]
Johannes Buchmann [2] Bart Preneel [1]

[1]K.U.Leuven

[2]TU Darmstadt

SHARCS - 10 September 2009

Why GPU?

(Source: MSI)

CUDA framework

Warning: sales talk

Your own personal supercomputer for $<$ €500.

Nvidia CUDA Framework:

- Run 'general' programs on GPU
- More complex operations, data types, branching...
- Recent GPU required
- Theory: 1TFlop (practice: 200 GFlop)

Crypto on GPU

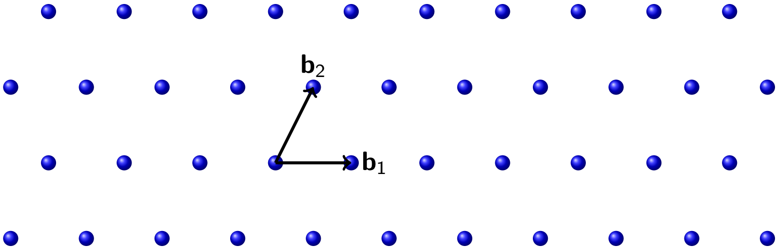
Current applications:

- Ciphers:
  - RSA [1], ECC [2], AES [3]
- Cryptanalysis:
  - Factoring [4]
  - Brute force

Focus: high throughput, not latency

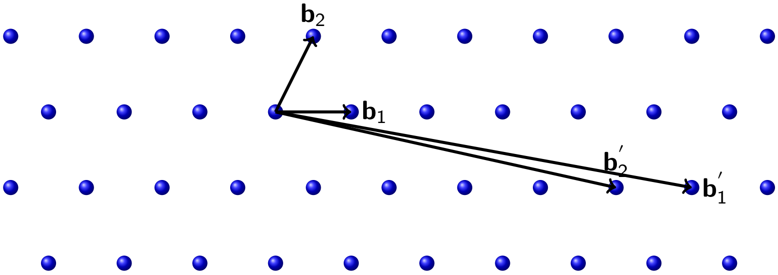
[1] Moss, Page, Smart / Szerwinski, Guneysu / Fleissner
[2] Szerwinski, Guneysu
[3] Manavski / Harrison, Waldron
[4] Bernstein, Chen, Cheng, Lange, Yang

Lattices



- Basis matrix $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ with $\mathbf{b}_i \in \mathbb{R}^d$
- Lattice: $L(\mathbf{B}) = \{\sum_{i=1}^{n} x_i \mathbf{b}_i, x_i \in \mathbb{Z}\}$

Shortest Vector Problem (SVP)



- Basis not unique
- Idea: 'good' basis $\mathbf{B}$ and 'bad' basis $\mathbf{B}'$
- Finding $\lambda_1(L)$ is hard with $\mathbf{B}'$

Algorithms for SVP

Shortest vector problem

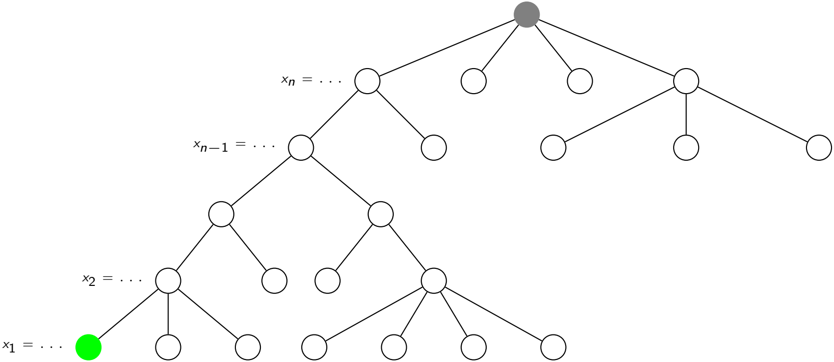
Compute $\min_{\mathbf{x} \in \mathbb{Z}^n} \|\mathbf{Bx}\|_2$

SVP algorithms:

- LLL (+variants): approximate solution, polynomial
- BKZ
- ...
- Enum: exact solution, exponential

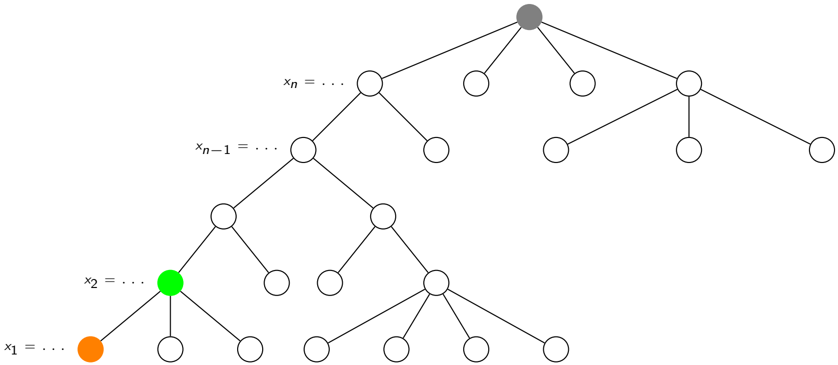
$\Longrightarrow$ This talk: focus on enum.

Enumeration



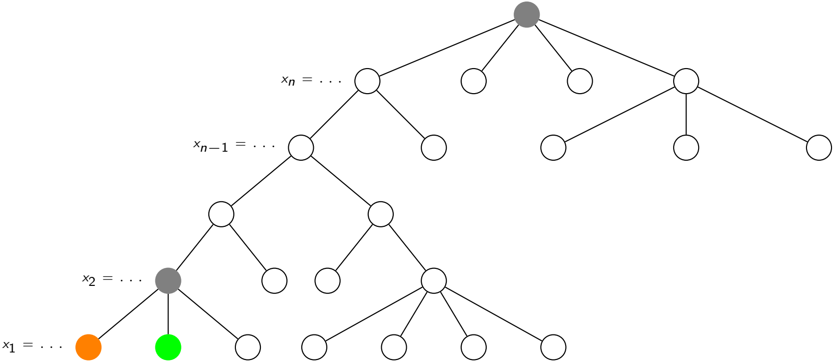
Optimum $A = \|\mathbf{Bx}\|_2^2$ and $\mathbf{x} = [1, 0, \dots, 0]$

Enumeration



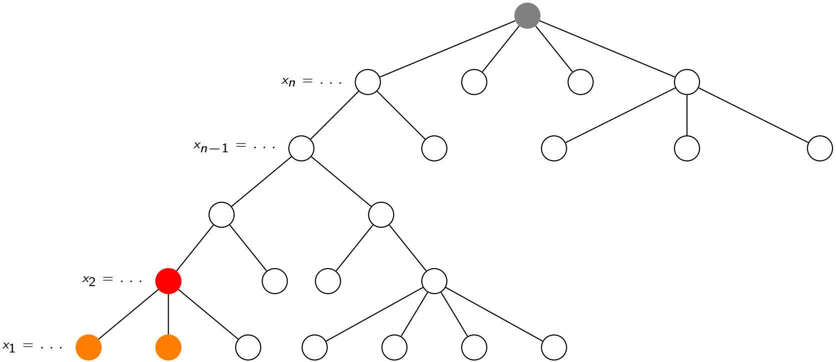
Intermediate norm l_2 s.t. $l_i \geq l_{i+1}$ (with $l_1 = \|\mathbf{Bx}\|_2^2$)

Enumeration



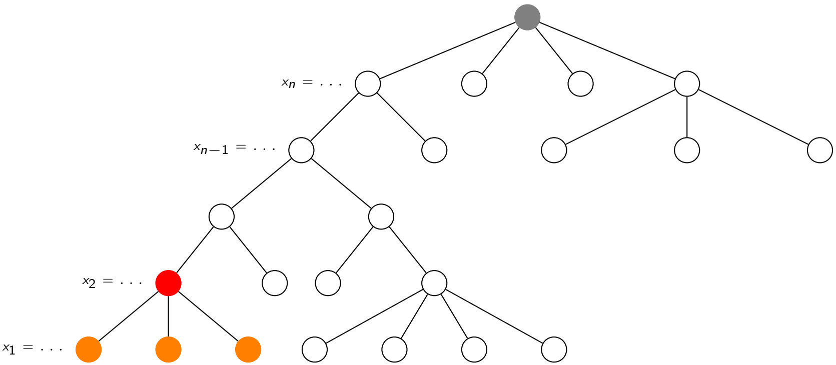
New optimum $A = \|\mathbf{Bx}\|_2^2$

Enumeration

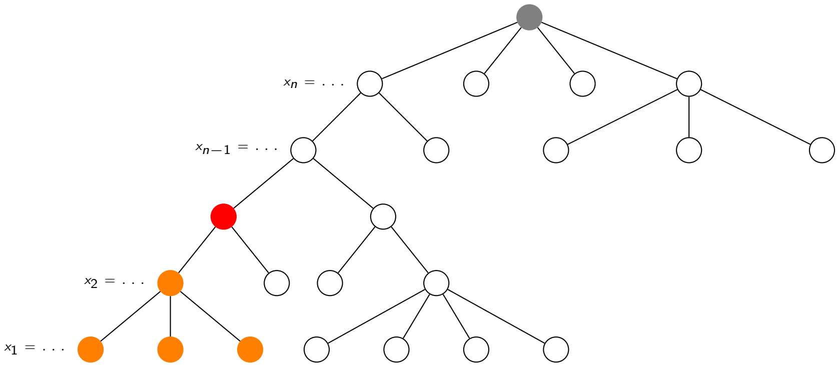


Cut off branch if $l_i > A$.

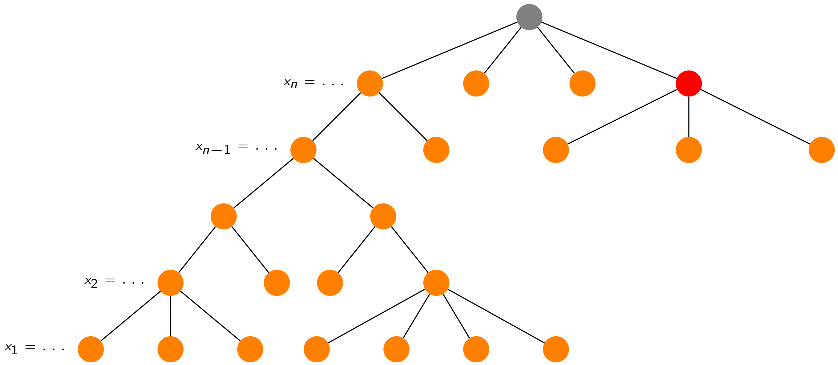
Enumeration



Enumeration



Enumeration

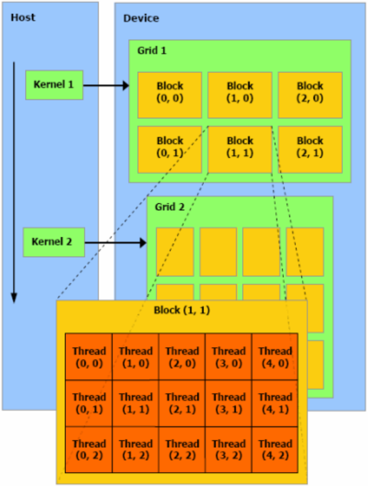


Processor

Nvidia GTX280:

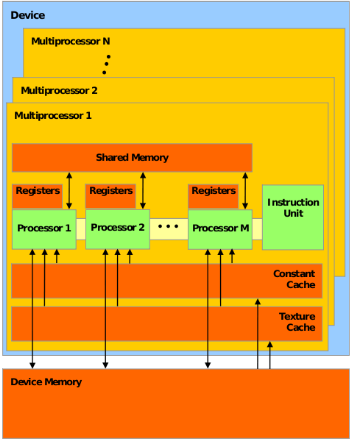
- 240 cores, scalar processors
- 30 multiprocessors (8 cores each)
- 1.3 GHz
- 1GB Global Memory
- 32 & 64-bit integers, FP

Programming model



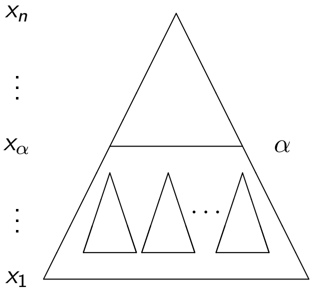
(Source: CUDA programming guide)

Memory types



(Source: CUDA programming guide)

Algorithm Flow



Basic idea

Input: $\mathbf{B}$, A, α, n

1 Compute the Gram-Schmidt decomposition of $\mathbf{b}_i$

2

3

Output: $(x_1, \ldots, x_n)$ with $\left\|\sum_{i=1}^{n} x_i \mathbf{b}_i\right\| = \lambda_1(L)$

Basic idea

Input: $\mathbf{B}$, A, α, n

1 Compute the Gram-Schmidt decomposition of $\mathbf{b}_i$
2 CPU: generate $\mathbf{x}_i = [0, \ldots, 0, x_\alpha, \ldots, x_n]$
3

Output: $(x_1, \ldots, x_n)$ with $\left\| \sum_{i=1}^{n} x_i \mathbf{b}_i \right\| = \lambda_1(L)$

Basic idea

Input: $\mathbf{B}$, A, α, n

1 Compute the Gram-Schmidt decomposition of $\mathbf{b}_i$
2 CPU: generate $\mathbf{x}_i = [0, \ldots, 0, x_\alpha, \ldots, x_n]$
3 GPU thread: run a sub-enum on $\mathbf{x}_i$, if new optimum $\rightarrow$ store in $\mathbf{x}$

Output: $(x_1, \ldots, x_n)$ with $\left\|\sum_{i=1}^{n} x_i \mathbf{b}_i\right\| = \lambda_1(L)$

Basic idea

Input: $\mathbf{B}$, A, α, n

1. Compute the Gram-Schmidt decomposition of $\mathbf{b}_i$
2. CPU: generate $\mathbf{x}_i = [0, \ldots, 0, x_\alpha, \ldots, x_n]$
3. GPU thread: run a sub-enum on $\mathbf{x}_i$, if new optimum $\rightarrow$ store in $\mathbf{x}$

Output: $(x_1, \ldots, x_n)$ with $\left\|\sum_{i=1}^{n} x_i \mathbf{b}_i\right\| = \lambda_1(L)$

$\Longrightarrow$ horrible performance

Early termination...

Input: $\mathbf{B}$, A, α, n

Compute the Gram-Schmidt decomposition of $\mathbf{b}_i$
CPU: generate $\mathbf{x}_i = [0, \ldots, 0, x_\alpha, \ldots, x_n]$

Output: $(x_1, \ldots, x_n)$ with $\left\| \sum_{i=1}^{n} x_i \mathbf{b}_i \right\| = \lambda_1(L)$

Early termination...

Input: $\mathbf{B}$, A, α, n

Compute the Gram-Schmidt decomposition of $\mathbf{b}_i$
CPU: generate $\mathbf{x}_i = [0, \ldots, 0, x_\alpha, \ldots, x_n]$
GPU thread:
while *there are* $\mathbf{x}_i$ *left.* **do**
  Start enum for a certain $\mathbf{x}_i = [0, \ldots, 0, x_\alpha, \ldots, x_n]$
  Stop enum after $\mathbf{S}$ steps, store the state $\{l_i, \bar{\mathbf{x}}_i, s_i = \mathbf{S}\}$
end

Output: $(x_1, \ldots, x_n)$ with $\left\|\sum_{i=1}^{n} x_i \mathbf{b}_i\right\| = \lambda_1(L)$

Early termination...

Input: $\mathbf{B}$, A, α, n

```
Compute the Gram-Schmidt decomposition of b_i
CPU: generate x_i = [0, ..., 0, x_α, ..., x_n]
GPU thread:
while there are x_i left. do
    Start enum for a certain x_i = [0, ..., 0, x_α, ..., x_n]
    Stop enum after S steps, store the state {l_i, x̄_i, s_i = S}
end
CPU: Get enum state x̄_i = [x̄_1, ..., x̄_{α−1}, x_α, ..., x_n]
```

Compute the Gram-Schmidt decomposition of $\mathbf{b}_i$

CPU: generate $\mathbf{x}_i = [0, \ldots, 0, x_\alpha, \ldots, x_n]$

GPU thread:

while *there are* $\mathbf{x}_i$ *left.* **do**

  Start enum for a certain $\mathbf{x}_i = [0, \ldots, 0, x_\alpha, \ldots, x_n]$

  Stop enum after $\mathbf{S}$ steps, store the state $\{l_i, \bar{\mathbf{x}}_i, s_i = \mathbf{S}\}$

end

CPU: Get enum state $\bar{\mathbf{x}}_i = [\bar{x}_1, \ldots, \bar{x}_{\alpha-1}, x_\alpha, \ldots, x_n]$

Output: $(x_1, \ldots, x_n)$ with $\left\|\sum_{i=1}^{n} x_i \mathbf{b}_i\right\| = \lambda_1(L)$

Early termination...

Input: $\mathbf{B}$, A, α, n

Compute the Gram-Schmidt decomposition of $\mathbf{b}_i$
CPU: generate $\mathbf{x}_i = [0, \ldots, 0, x_\alpha, \ldots, x_n]$
GPU thread:
while *there are* $\mathbf{x}_i$ *left.* **do**
    Start enum for a certain $\mathbf{x}_i = [0, \ldots, 0, x_\alpha, \ldots, x_n]$
    Stop enum after $\mathbf{S}$ steps, store the state $\{l_i, \bar{\mathbf{x}}_i, s_i = \mathbf{S}\}$
end
CPU: Get enum state $\bar{\mathbf{x}}_i = [\bar{x}_1, \ldots, \bar{x}_{\alpha-1}, x_\alpha, \ldots, x_n]$
CPU: Continue enum if $\mathbf{S}$ was reached

Output: $(x_1, \ldots, x_n)$ with $\left\| \sum_{i=1}^{n} x_i \mathbf{b}_i \right\| = \lambda_1(L)$

$\Longrightarrow$ solves length difference problem, still not so good

Iterating

Input: $\mathbf{B}$, A, α, n

```
Compute the Gram-Schmidt decomposition of b_i
while true do
    CPU: generate some x_i = [0, ..., 0, x_α, ..., x_n]

    CPU: Get enum state x̄_i = [x̄_1, ..., x̄_{α-1}, x_α, ..., x_n]
end
```

Compute the Gram-Schmidt decomposition of $\mathbf{b}_i$

while *true* **do**

CPU: generate some $\mathbf{x}_i = [0, \ldots, 0, x_\alpha, \ldots, x_n]$

CPU: Get enum state $\bar{\mathbf{x}}_i = [\bar{x}_1, \ldots, \bar{x}_{\alpha-1}, x_\alpha, \ldots, x_n]$

end

Output: $(x_1, \ldots, x_n)$ with $\left\| \sum_{i=1}^{n} x_i \mathbf{b}_i \right\| = \lambda_1(L)$

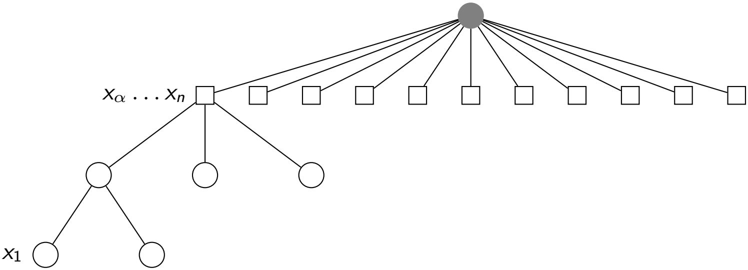
Iterating

Input: $\mathbf{B}$, A, α, n

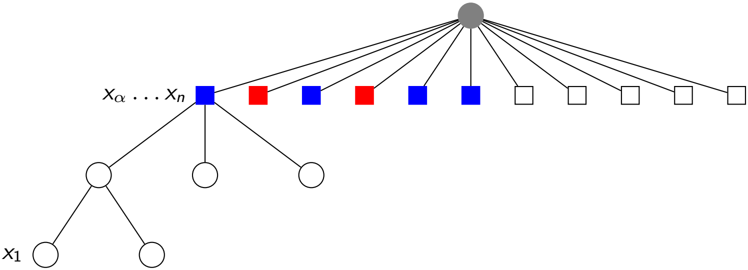
```
Compute the Gram-Schmidt decomposition of b_i
while true do
    CPU: generate some x_i = [0, ..., 0, x_α, ..., x_n]
    GPU thread:
    while there are x_i left. do
        Start enum for a certain x_i or continue enum for x̄_i
        Stop enum after S steps, store the state {l_i, x̄_i, s_i = S}
    end
    CPU: Get enum state x̄_i = [x̄_1, ..., x̄_{α-1}, x_α, ..., x_n]
end
```

Output: $(x_1, \ldots, x_n)$ with $\left\|\sum_{i=1}^{n} x_i \mathbf{b}_i\right\| = \lambda_1(L)$

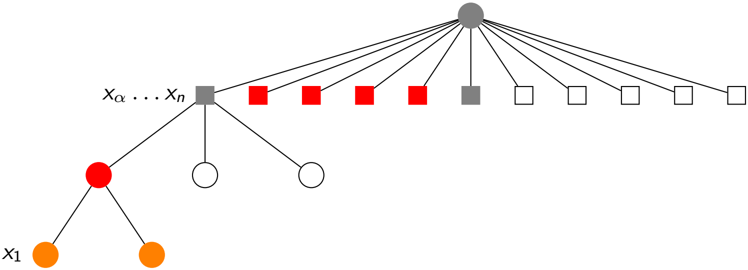
GPU Enumeration



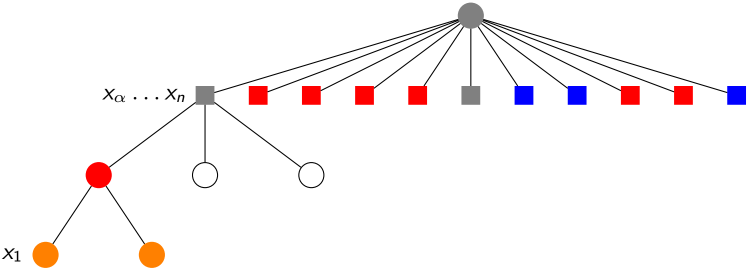
GPU Enumeration



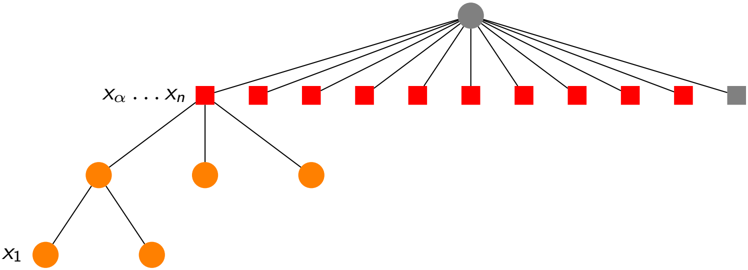
GPU Enumeration



GPU Enumeration



GPU Enumeration



Implementation details

Some facts & figures:

- Dimension 50, 100000 starting vectors $\rightarrow$ upload & download $\sim$ 20 MB of data to GPU
- CPU top enum: very fast (low dimension)
- GPU runs for $>$ 10 seconds per iteration, iteration overhead is limited
- Share new optimal values among GPU threads

Throughput

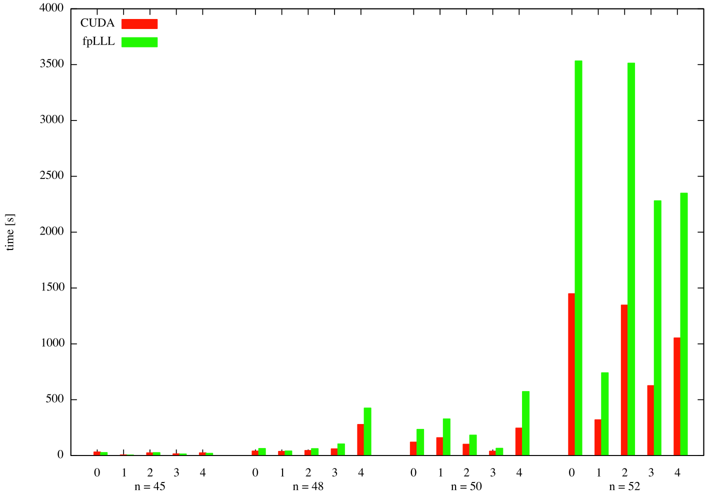
Throughput:

- CPU: around $25 \cdot 10^6$ steps/s
- GPU: up to $100 \cdot 10^6$ steps/s

Throughput on GPU depends on:

- Lattice dimension n
- Length of sub-enumerations
- Number of parallel threads, uploaded points...

Results


4000
3500
3000
2500
2000
1500
1000
500
0
time [s]
CUDA
fpLLL
0 1 2 3 4
n = 45
0 1 2 3 4
n = 48
0 1 2 3 4
n = 50
0 1 2 3 4
n = 52

Results

n	45	48	50	52	54
fpLLL	18.3s	139s	277s	2483s	6960s
CUDA	20.2s	92s	133s	959s	2599s
	110%	66%	48%	39%	37%

Table: Average time needed for enumeration of lattices in each dimension n.

Ideas for the Future

Future:

- Generalize ideas (not specific for gpu's... clusters?)
- Use full power of CPU (now: idle during gpu-time)
- Gaussian heuristic

The end...

Questions?

Algorithm

Algorithm 1: High-level GPU ENUM Algorithm

Input: $\mathbf{b}_i$, A, α, n

```
Compute the Gram-Schmidt decomposition of b_i
while true do
    S = {(x_i, Δx_i, Δ^2 x_i, l_i = α, s_i = 0)}_i ← Top enum: generate at most NUMSTARTPOINTS − #T vectors
    R = {(x̄_i, Δx_i, Δ^2 x_i, l_i, s_i)}_i ← GPU enumeration, starting from S ∪ T
    T ← {R_i : s_i ≥ S}
    if #T < CPUTHRESHOLD then
        Enumerate the starting points in T on the CPU.
        Stop
    end
end
```

Output: $(x_1, \ldots, x_n)$ with $\left\|\sum_{i=1}^{n} x_i \mathbf{b}_i\right\| = \lambda_1(L)$