upro EU
PROMOTIONAL PARTNERSHIP

Improving Data-Parallel Construction of Dn-Nets with Maximum Dispersion

Peter Zinterhof, Dept. of Scientific Computing, University of Salzburg

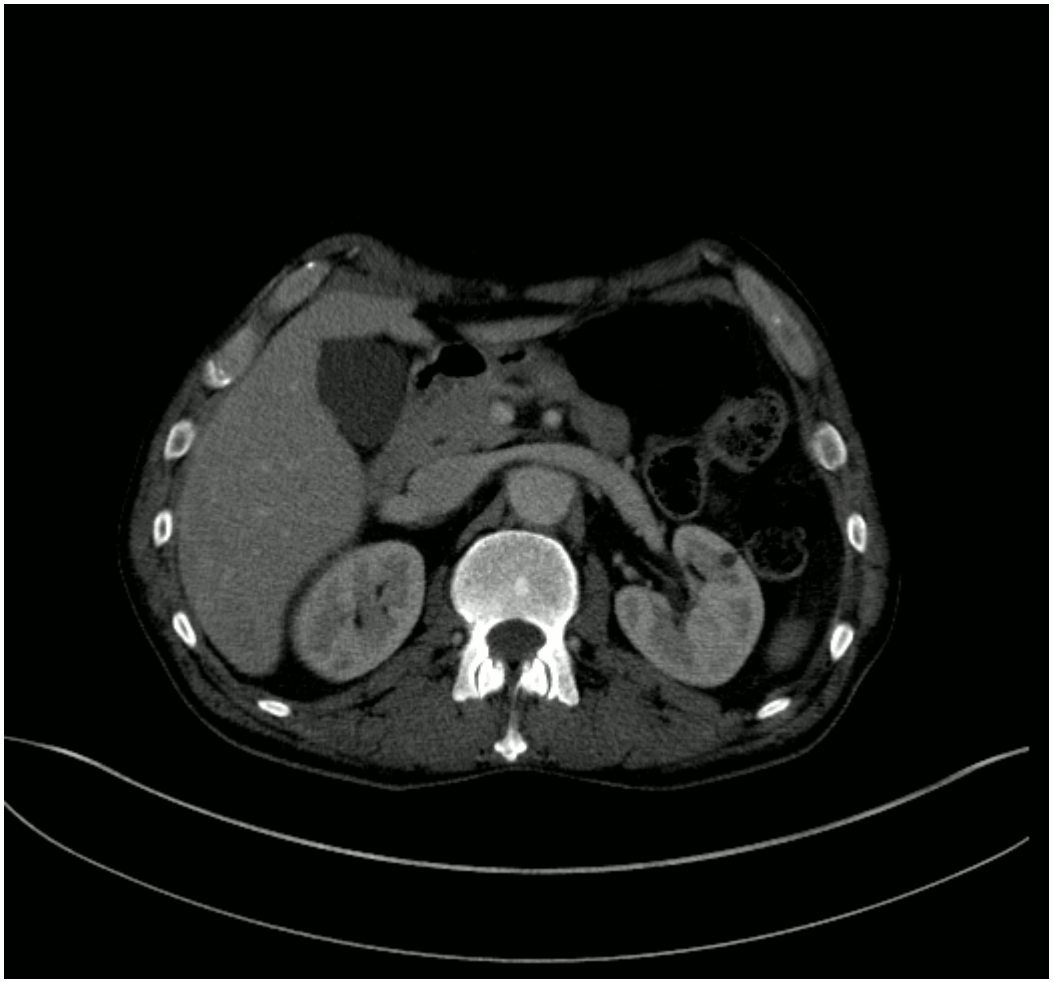
peter.zinterhof3@sbg.ac.at

Data-parallel construction of Dn-Nets

Motivation
Joint project: SALK (Salzburger Uniklinik)
Computed Tomography (CAT scans)

Task: Image Segmentation
-> detection of cysts in the body

Sample CAT-scan:



~ 500 patients , 200 – 300 slices per patient
Trainings set: ~ 140.000 slices

Ground truth:
Kidneys + cysts marked manually

CAT scan segmentation

Principal Component Analysis
algorithmic creation of 64x64 pixel-patches
auto-correlation matrix
Eigensystem -> preserve 32 most emminent vectors

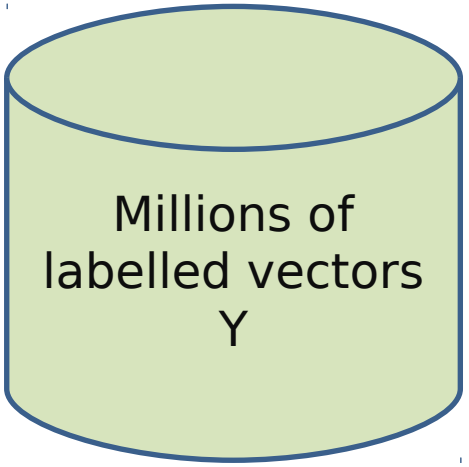
Code book
10 – 20 mio. Entries (vectors of 32 floats+labelling data)

Image Segmentation:

Convolution of new patient‘s CAT scan book vectors

~ 50.000 vectors X per slice

Code



Task: For each new patch X find nearest neighbor in Y and assign corresponding label Y_label.

Speeding up the Process:

Options:

Parallel and distributed computing (clusters)

Novel hardware -> GPGPU, CELL BE, FPGA

Speedup the Process

Options:

Parallel and distributed computing (clusters)

Novel hardware -> GPGPU, CELL BE, FPGA

Reduction of code book size -> reduce workload per (kD-trees, sorting, etc. does not work -> curse of dimensionality)

Dn-Nets with Maximum Dispersion

Mathematical concept of Dn-Nets

Metric Dispersion

$$d(x,y) = (\sum_{k=1}^{s} |x_k - y_k|^2)^{1/2}$$

$$\delta_N = \delta(x_1, x_2, ..., x_N) = \max_{x \in E}(\min(d(x, x_k))$$

Dn-Net over finite metric space E approximates high-dimensional data up to some arbitrary error e

Simpler interpretation: Dn is the radius of the largest hyper-sphere, that fits within (x1,x2,.., xn).

Construction of Dn-Nets

Choose two seed points (x1, x2) with d(x1,x2) = max (d(dx, dy)) for all x,y in E

Iteratively add point x, such that max(min(d(x,xk),k=1,..,N) is reached.

In other words: at each step we add that point x to our Dn-Net, which decreases the value of Dn by the smallest possible amount.

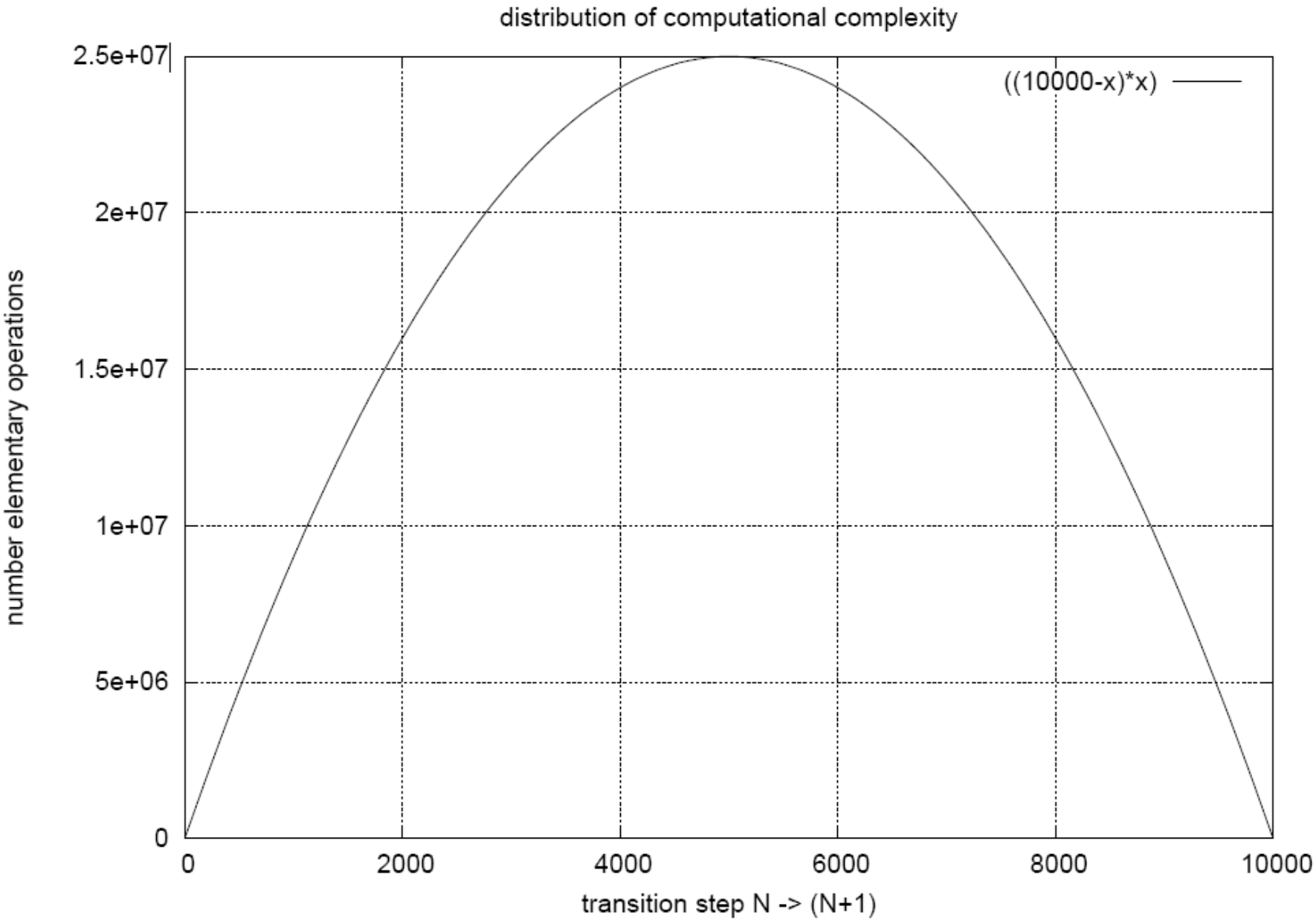
Previous work

High computational complexity

Transition from D(n) -> D(n+1): (E-N)N elementary operations

Total complexity: O(E N^2- N^3)

Elementary operation: euclidean distance d(x,y)

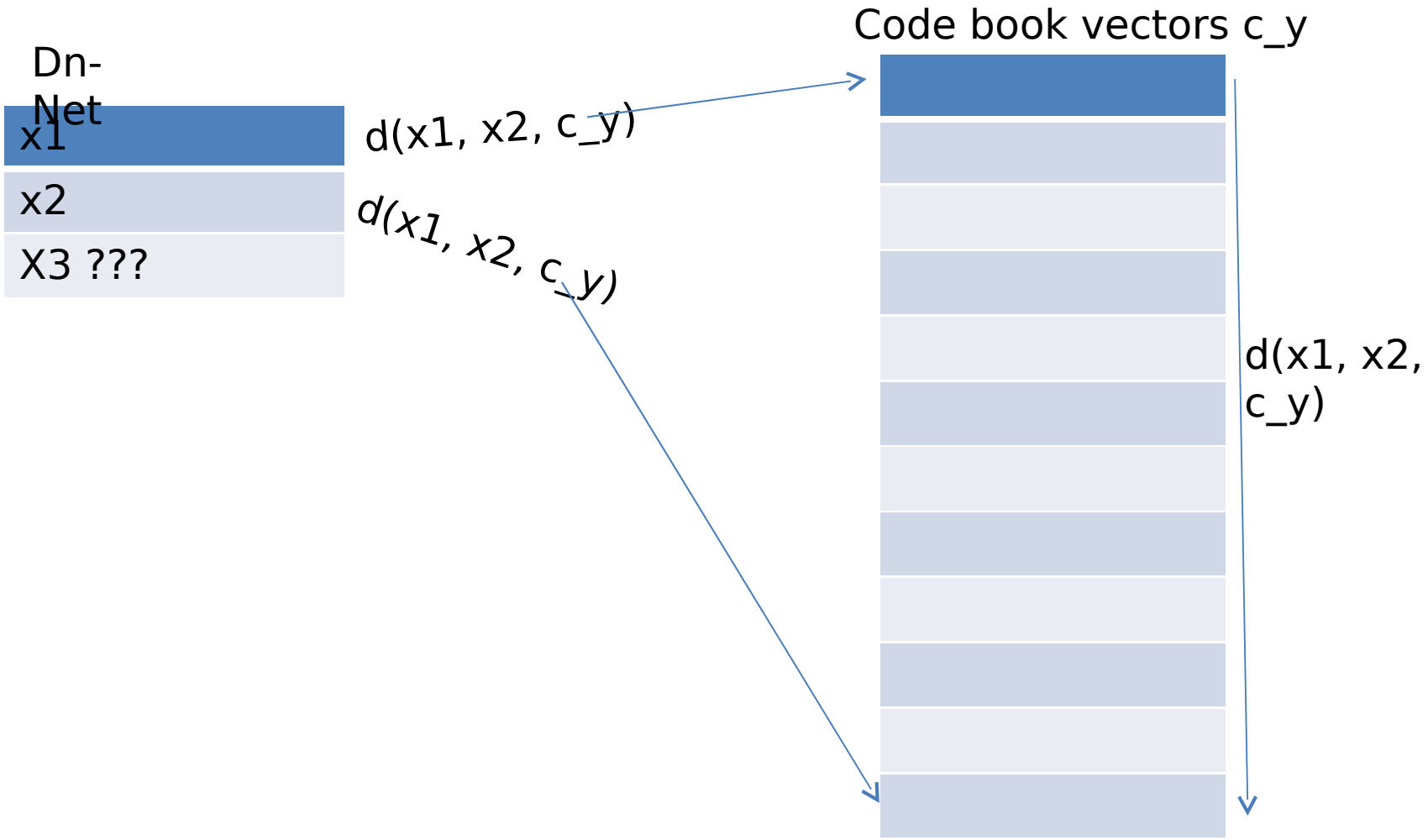

distribution of computational complexity
((10000-x)*x)
number elementary operations
2.5e+07
2e+07
1.5e+07
1e+07
5e+06
0
0
2000
4000
6000
8000
10000
transition step N -> (N+1)

Improved construction of Dn-Nets

Basic idea: instead of (re)computing distances between potential follow-up vectors Xk for a given Dn-Net, the minimum distances for each vector in relation to the Dn-Net are kept in memory.

Also: in the previous algorithm the order in which vectors were added to the Dn-Net had been omitted -> we had to compute distances (d(DN, xk)) over and over again for each new vector.

Previous vs. Improved Algorithm I



Previous vs. Improved Algorithm II

Dn-Net

x1
x2
X3 ???

Distances array D

d(x1, c1)
d(x1, c2)
d(x1, c3)
d(x1, c4)
....
d(x1, c12345678)

We KNOW x2 already !!!
(because it‘s the second of our two seed points)

stances array D holds the current minimum distances
tween Dn-Net and entries c1...cN. The only vector that
n change the whole picture is the follow-up-vector x2.
us we only have to compare x2 and the values of
x1,cx), no matter how big Dn has already grown.

Previous vs. Improved Algorithm III

x1
x2
X3 ???

Adding x2 involves a simple operation:

For all entries x of the codebook:

Step 1: Cx = min (**d(x1,c1)**, d (x2,c1)) with **d(x1,c1)** already stored in the array.

Step 2: find max(Cx) ‚on-the-fly‘ during the updates, which denotes the next follow-up vector x_(n+1). We can iteratively feed this vector into step 1 again and find all follow-up vectors in linear overall complexity.

d(x1, c1)
d(x1, c2)
d(x1, c3)
d(x1, c4)
....
d(x1, c12345678)

Data-Parallelism

al data-independance during update of Distances array D

Computations very well suited for GPGPUs: e.g. CUDA, OpenCL

plementation by means of two kernels:

- Find seed points x1, x2
  brute force: for each Cn find max(d(Cn, Cx)), x=(1..codebooksize)
  real world data: ~ 10 – 30 petaflops needed
- Update Distances D

Results

Computation of some small Dn-Nets on i7 and GPGPU

	i7-CPU, 2.66 GHz	**NVIDIA GTX680**	**NVIDIA GTX 480**
Δ_N-net:50k vectors, E=100k vectors	543.94 1.0x	32.70 16.6x	51.75 10.5x
Δ_N-net :500k vectors, E=1m vectors	54433.2 1.0x	2642.84 20.5x	2856.46 19.0x
Δ_N-net:5500 vectors, E=6.3 mio. vectors	7852.6 1.0x	185.67 42.2x	243.97 32.1x
Δ_N-net:15000 vectors, E=6.3 mio. vectors	21392.8 1.0x	499.56 42.8x	652.86 32.7x

Conclusions

esented an improved algorithm for the construction of Dn-Nets, which results
tal computational complexity of O(En^2) and linear complexity O(E) in each ste

ew algorithm can be considered a big improvement as it enables the constructio
DN-Nets that are common in real-world problems, and which could not be tackle
revious methods even on large clusters of GPGPU-systems.

resented algorithm lends itself well for GPGPU-execution, which typically enables
onal speedups in the range of 30-40x against single cpu cores and 10x against
core CPUs.

Conclusions

esented an improved algorithm for the construction of Dn-Nets, which results tal computational complexity of O(En^2) and linear complexity O(E) in each ste

ew algorithm can be considered a big improvement as it enables the constructio DN-Nets that are common in real-world problems, and which could not be tackle revious methods even on large clusters of GPGPU-systems.

resented algorithm lends itself well for GPGPU-execution, which typically enables onal speedups in the range of 30-40x against single cpu cores and 10x against core CPUs

Thank You !

UNIVERSITÄT SALZBURG