GPU Parallel Implementation of The Approximate K-SVD Algorithm Using OpenCL

Paul Irofti[1] Bogdan Dumitrescu[2]

[1]University Politehnica of Bucharest
paul@irofti.net

[2]Tampere University of Technology
bogdan.dumitrescu@tut.fi

EUSIPCO'2014

Outline

1. Introduction
2. OpenCL
3. AK-SVD
4. PAK-SVD
5. Conclusions

The problem

Given:

- initial dictionary D_0
- set of training signals Y
- target sparsity s
- number of iterations K

Output:

- trained dictionary D
- sparse representations X

Such that $Y \approx DX$.

Optimization Problem

Solving the optimization problem of:

$$\begin{aligned} \underset{D,X}{\text{minimize}} \quad & \|Y - DX\|_F^2 \\ \text{subject to} \quad & \|x_i\|_0 \le s, \ \forall i \end{aligned}$$

General Approach

Most algorithm iterations involve two essential steps:

- **sparse coding** Y using dictionary D resulting X
- **updating the dictionary** using the current representations X

Existing solutions:

- Sparse representations:
  - SP
  - MP
  - OMP
- Dictionary update:
  - MOD
  - K-SVD
  - AK-SVD

Current State

Practical applications employing these methods

- show good results
- low representation errors
- slow running times
- top consumer: the sparse representation stage
- dictionary update performed one atom at a time
- each update step depends on the one before it

Our approach:

- update more than one atoms at a time
- distributed sparse coding
- new parallel algorithm PAK-SVD

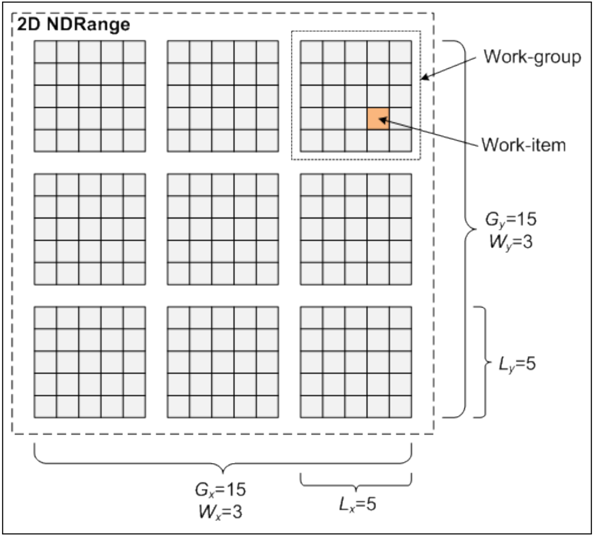
Platform

OpenCL platform

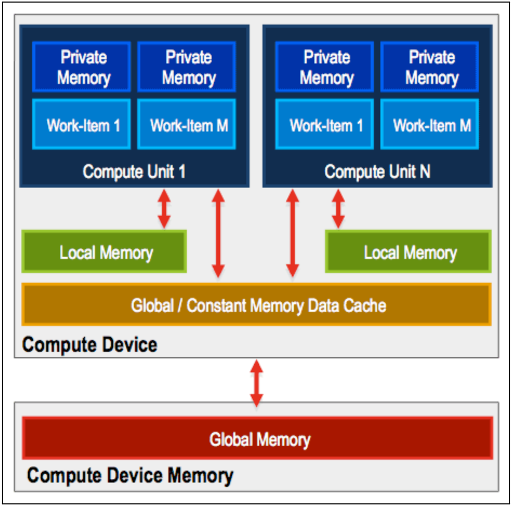
- execute small functions (kernels) in parallel
- processing elements $\subset$ compute units $\subset$ OpenCL device
- work load topology defined as an n-dimensional space

Notation: $NDR : \langle x, y, z \rangle$

N-Dimensional Range – 2D Example



Memory Layout



Hardware

ATI FirePro V8800 (FireGL V) specifications:

- 1600 streaming processors
- 2048MB global memory
- 32KB local memory
- 256 maximum work-group size
- 20 maximum compute units
- OpenCL v1.2 compliant
- 2640 single-precision GFLOPS
- 528 double-precision GFLOPS.

Time Counting

Counting in CPU ticks bypassing:

- unsynchronized tick counts between different cores on a multiprocessor system
- lack of serialization with MSVC compilers on x64 systems
- EBX/RBX register spilling issues with GCC compilers when using position independent code

On the machine we tested one tick represents roughly 0.3125ns.

AK-SVD Algorithm

Data:

- given dictionary D and signal set Y
- compute sparse representations X and optimize dictionary D

Iterations:

- **sparse coding:** for each signal y in Y
  - use $\mathrm{OMP}(D, y)$ for representing x of X
- **dictionary update:** for each atom d in D
  - remove d from the dictionary
  - find the singals using d in their representation
  - optimize d keeping the representations and the dictionary fixed
  - update the representations by using the new atom d
  - update the dictionary by reintroducing the optimized atom d

Comments

Observations:

- the dictionary is changed on each update step
- so are the sparse representations
- the current atom's update **depends on all** of the atoms updated before it
- AK-SVD eliminates the need to explicitly compute the residual

PAK-SVD Sparse Coding

Data:

- given dictionary $D \in \mathbb{R}^{p \times n}$ and signal set $Y \in \mathbb{R}^{p \times m}$
- compute sparse representations $X \in \mathbb{R}^{n \times m}$

Sparse Coding with OMP:

- using an NDR($\langle m \rangle$, $\langle any \rangle$) splitting
- big memory foot-print $O(ns)$, where s is the desired sparsity
- all the matrices are kept in global memory
- each PE computes OMP for a single data item from Y

PE_1 $X_1 = \text{OMP}(Y_1)$	PE_2 $X_2 = \text{OMP}(Y_2)$	$\dots$	PE_m $X_m = \text{OMP}(Y_m)$

PAK-SVD Dictionary Update

Data:

- $D \in \mathbb{R}^{p \times n}$, $Y \in \mathbb{R}^{p \times m}$ and $X \in \mathbb{R}^{n \times m}$

Dictionary update for **batches of $\tilde{n}$ atoms** from D:

- calculate the full residual matrix $E = Y - DX$
- for each atom from the current batch do in **parallel**
  - compensate the error matrix E as if the current atom was missing from the dictionary
  - find the singals using d in their representation
  - optimize d keeping the representations and the error matrix fixed
  - update the representations by using the new atom d
  - update the dictionary by reintroducing the optimized atom d

PAK-SVD Dictionary Update (2)

We use an NDR($\langle\tilde{n}\rangle$, $\langle any\rangle$) splitting for updating $\tilde{n}$ atoms at a time:

PE_1 D_1, X_{D_1}	PE_2 D_1, X_{D_2}	$\ldots$	$PE_{\tilde{n}}$ $D_{\tilde{n}}$, $X_{D_{\tilde{n}}}$

Each PE is in charge of updating one atom. Memory layout:

- private: d, the atom being updated
- local or global: $\mathcal{I}$, indices of signals using d
- global: E, X, D

Matrix Multiplication

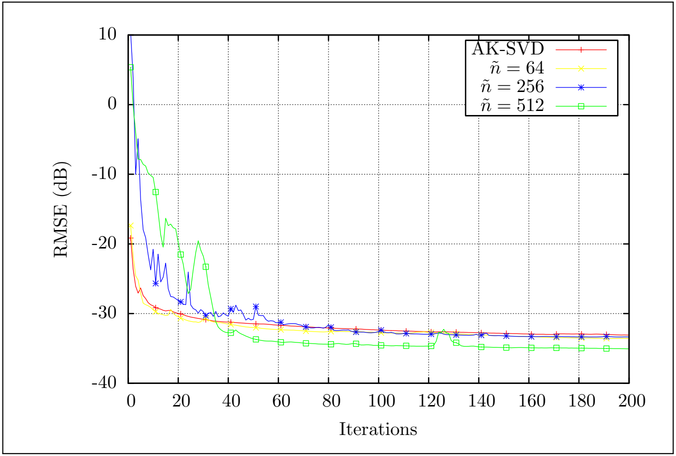
OpenCL implementation:

- split the N-dimensional space as NDR$(\langle n, m\rangle, \langle 64, 64\rangle)$
- block-based multiplication
- calculating a block is performed within a work-group

Memory layout:

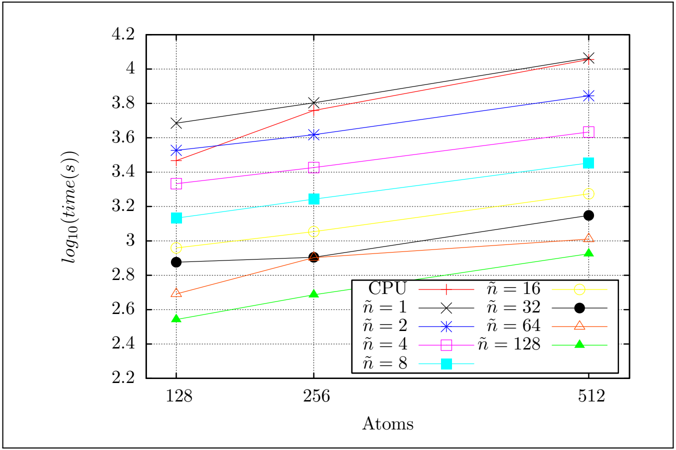
- global: input and output matrices
- local: copied input block sub-matrices
- private: vectorized types for dot operations

Error



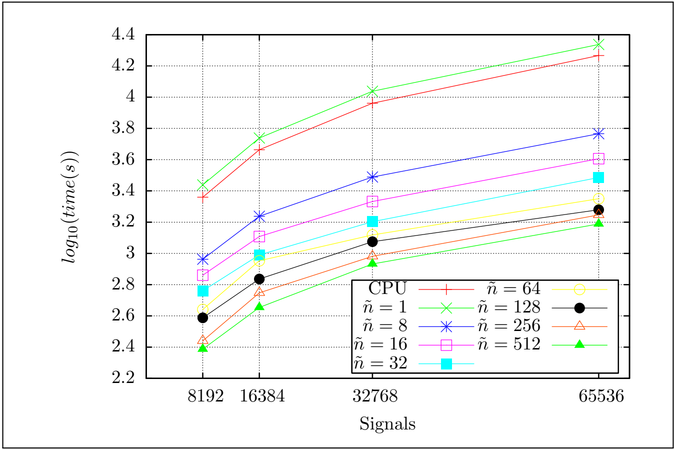
Error evolution for $m = 16384$, $n = 512$, $s = 12$.

Performance (1)



Execution times for $m = 16384$, $s = 10$, $K = 200$.

Performance (2)



Execution times for $n = 512$, $s = 8$, $K = 100$.

More Error Results

Table: Final errors for AK-SVD and PAK-SVD with $\tilde{n} = n$.

		n					
		128		256		512	
		AK	PAK	AK	PAK	AK	PAK
	4	0.0425	0.0407	0.0385	0.0387	0.0376	0.0372
	6	0.0374	0.0349	0.0334	0.0316	0.0311	0.0297
s	8	0.0345	0.0306	0.0294	0.0272	0.0259	0.0245
	10	0.0322	0.0276	0.0276	0.0239	0.0233	0.0206
	12	0.0319	0.0249	0.0254	0.0205	0.0221	0.0176

Conclusions

PAK-SVD improves AK-SVD:

- performs up to 12x faster
- parallel sparse coding stage
- **parallel dictionary update**
- **smaller representation error**