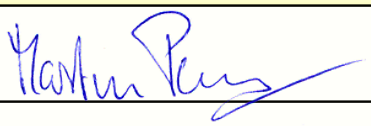
Parallelised Data-Mining Components

Executive Summary

	Name	*Signature*	*Date*
Prepared by:	Martin Pačes / ISS		19/01/2012
Approved by:			
Accepted by:			

Distribution List

Name	*Company / Agency*	*E-mail*
Sergio D'Elia	ESA	Sergio.DElia@esa.int
Alex Madden	ESA	Alex.Madden@esa.int
Michele Iapaolo	ESA	Michele.Iapaolo@esa.int
Pier Giorgio Marchetti	ESA	Pier.Giorgio.Marchetti@esa.int
Jordi Farres	ESA	Jordi.Farres@esa.int
Martin Pačes	ISS	Martin.Paces@iguassu.cz
Petr Bareš	ISS	Petr.Bares@iguassu.cz
Miroslav Houdek	ISS	Miroslav.Houdek@iguassu.cz

Changes

Issue	*Date*	*Change Record*	*Author*
1.0	10/01/2012	first version	Martin Pačes (ISS)
1.1	19/01/2012	Adding Tab. 1-4 containing benchmarking results of the implemented algorithms as agreed at the FP.	Martin Pačes (ISS)

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Scope of the Document

The document summarises the performed activities and achievements of the of the PalDMC project.

1.2 Applicable Documents

AD01 PalDMC AO6025 Proposal, v.2.3

AD02 PalDMC-ISS-CN01-001-MoM – Minutes of Contractual Negotiation Meeting 20/11/2009

AD03 PalDMC Contract, No. 22702/09/I-AM

AD04 PalDMC-ISS-KO-002-MoM – Minutes of Kick-Off Meeting 25/11/2009

AD05 PalDMC-ISS-PM-004-MoM – Minutes of Progress Meeting 25/08/2010

AD06 PalDMC-ISS-CDR-005-MoM – Minutes of CRD 19/11/2010

AD07 PalDMC-ISS-PM2-006-MoM – Minutes of Progress Meeting 26/01/2011

AD08 PalDMC-ISS-PM3-007-MoM – Minutes of Progress Meeting 26/01/2011

AD09 PalDMC-ISS-PM4-008-MoM – Minutes of Progress Meeting 28/04/2011

AD10 PalDMC-ISS-PM4b-009-MoM – Minutes of Progress Meeting 14/09/2011

AD11 PalDMC-ISS-PM5-010-MoM – Minutes of Progress Meeting 03/10/2011

AD12 Confidentiality Agreement between MEEO and ISS, 22/02/2011

1.3 Reference Documents

RD01 PalDMC-ISS-003-PDD PalDMC Prototype Design Document,

RD02 PalDMC-ISS-012-TN Road Extraction - Test Report

RD03 http://www.orfeo-toolbox.org/otb

2 PalDMC Project

The prime objective of the PalDMC (Parallel Data Mining Components) project carried out by consortium of Iguassu Software Systems, a.s., (ISS) and Advanced Computer Systems, Spa., (ACS) was to design and implement performance optimised versions of selected core Data Mining algorithms to be used by Earth Observation (EO) Image Information Mining (IIM).

The IIM, as the name suggests, is a set of techniques used to exploit, as much efficiently as possible, meaningful information contained by images. In order to obtain semantic content of an image sophisticated algorithms with high processing cost must be usually applied. Beside this baseline processing cost, the analysis of EO data is further complicated by continually increasing data volumes thanks to the trend toward very high-resolution sensors. Yet, the amount of the processed data can increase furthermore when the image time series is considered, i.e., when several acquisition systems, large set of multi-band images, and multiple acquisitions times are analysed. In order to make these calculations feasible performance optimisation, including the parallel processing, must be inevitably taken into account, at least, for the most computationally expensive 'bottle-neck' algorithms involved in the data analysis.

The activities of the PalDMC projects were split in two consecutive parts. During the first, design part, a study on selection of the best SW techniques and HW selection has been carried out. A single algorithm has been selected (K-means clustering) and prototyped using different combinations of parallel processing such as the SIMD (Single Instruction Multiple Data) CPU instructions, multi-threading on multi core-CPUs, heterogeneous GP-GPU (General Purpose Graphical Processing Unit) computing, distributed computing on network interconnected computers. The results are documented in-detail in [RD01]. Although the original project proposal focused primarily on large scale distributed computing [AD01], on ESA request it was refocused on GP-GPU computing [AD02, AD04, and AD07].

The second part of the projects focused on implementation on the performance optimised algorithms. The following algorithms were selected for implemented:

- **K-means** (Generalized L'loyd Algorithm and Linde-Brutzo-Gray variants; GP-GPU, multi-threaded + SIMD instructions implementations.)
- **Pan-Sharpening** (Simple Orfeo-Toolbox (OTB) [RD03] pan-sharpening and Least Square fitting variants, multi-threaded implementation.)
- **MEEO Segmentation** (algorithm provided by MEEO Srl. [AD12], multi-threaded implementation)
- **Road Extraction** (Orfeo-Toolbox (OTB) [RD03] road extraction algorithm, Orfeo-Tolbox Link-Path algorithm performance improving patch and multi-threaded and GP-GPU reimplementation of the raster processing part of the OTB algorithm.

The particular activities are discussed more in detail in following sections.

2.1 Study on Algorithm Performance Optimisation and Parallel Computing

As part of the prototype design, a thorough study [RD01] on the subject of performance optimisation and parallel and distributed computing were performed, supported by the review of the information published elsewhere and by the results of rapid prototyping as prove of concept.

For the purpose of the rapid prototyping we had chosen the K-means algorithm [RD01] since this

algorithm:

1) is well known and trivial to implement;
2) belongs to a class of resource demanding algorithms with a higher computing complexity

$$O(N_K, N_S, N_D, N_I) = N_K \times N_S \times N_D \times N_I \quad ,$$

where N_K, N_S N_D, and N_I are the number clusters' centroids, vector samples, feature space dimension and number of iterations, respectively (Euclidean distance is used as the similarity measure);

3) has large amount of exploitable parallelism;
4) is one of the key algorithms used by the KEO image information system during the initial feature extraction phase and it posed performance bottle-neck.

Different options of parallel computing were examined. For details on SW and HW arrangement see [RD01]. These were:

- reference sequential plain C++ implementation,
- multi-threaded plain C++ implementation,
- single thread Intel SSE2 (SIMD CPU) vectorised implementation,
- multi-thread Intel SSE2 (SIMD CPU) vectorised implementation,
- NVidia CUDA (GP-GPU) implementation,
- distributed FastRPC+SSE2+multi-thread implementation,
- distributed FastRPC+CUDA implementation.

The *speed-up* (how much faster the parallel implementation runs faster than the reference sequential one) and *efficiency* (departure from ideal *speed-up* due to the overheads) were used as

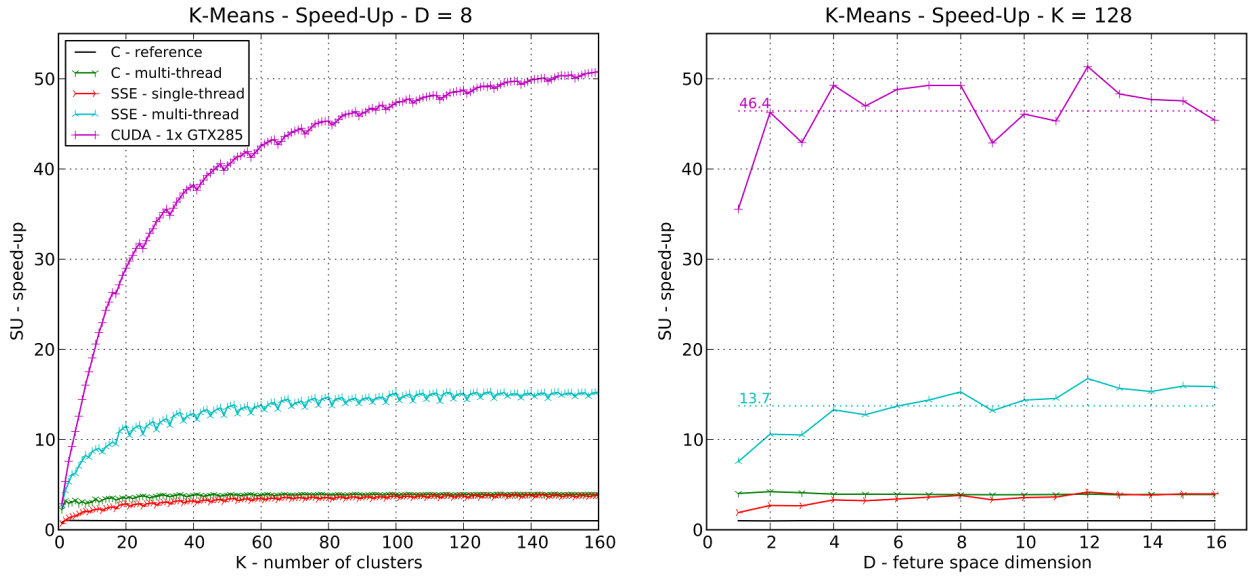


Figure 1: CPU local calculation: Measured speed-up of local (single computer) K-means calculations as function of the algorithms parameters: number of clusters (left; fixed feature space dimension) and feature space dimension (right; fixed number of clusters). Different options of parallel processing are compared: sequential SSE2 enabled calculation (red, SU→4), multi-threaded plain C++ implementation (green, SU~4), combined multi-threaded and SSE2 enabled implementation (cyan, SU→16), and GP-GPU implementation (magenta). The reference sequential plain C++ implementation is indicated (black line,SU = 1).

K-means Execution Time	*single iteration / s*	*50 iterations*	*network transfer / s*	*50 iter. worst case total*
C – single thread	51.7	43 min 3 s	---	43 min 15 s
C – multi-thread	12.4	10 min 19 s	---	10 min 31 s
SSE2 – single thread	13.5	11 min 18 s	---	11 min 30 s
SSE2 – multi-thread	3.5	2 min 53 s	---	3 min 5 s
CUDA – 1xGPU	1.1	55 s	---	1 min 7 s
CUDA – 2xGPU	0.6	30 s	---	42 s
FastRPC – 1xCPU	3.6	3 min 1 s	9.5	3 min 22 s
FastRPC – 2xCPU	1.8	1 min 32 s	7.0	1 min 51 s
FastRPC – 3xCPU	1.3	1 min 5 s	7.8	1 min 25 s
FastRPC – 1xGPU	1.2	1 min	9.7	1 min 22 s
FastRPC – 2xGPU	0.7	34 s	7.0	53 s
FastRPC – 3xGPU	0.5	25 s	6.8	44 s

Table 1: The average processing time of 50 iteration k-means algorithms for 1GiB of processed data (cube of 4096x8192x8 32bit FP numbers) and 128 clusters. For network distributed implementation of the k-means algorithm the times needed to distribute the processed data to the worker nodes are also presented. The 'worst-case' total execution time is a sum of the fixed overhead, the possible initial data distribution and the time spent by 50 k-means iterations. The minimal and maximal execution times are highlighted. The presented values are calculated as an average value of three independent measurements. HW configuration: AMD Phenom II X4 @ 3.0 GHz, 4 GiB DDR3 @ 1.33 Ghz, 2x NVidia GTX 285 2 GiB DDR2, 2x 256 GB SATA II SW-RAID0.

measure of the performance improvement.

The tests' results of the K-means algorithm led us to following conclusions. The best options in terms of the performance is the use of multiple GP-GPU cards on a single computer. In case of distributed GP-GPU calculation the good performance can be obtained, but the overhead due to the network (1Gps Ethernet) data transfers significantly reduces the efficiency of the very fast calculation and thus high speed interconnects (e.g., 10Gps Ethernet) shall be used.

In case of pure CPU calculations the combination of SSE SIMD instruction and multi-threading may provide satisfactory performance boost. The obtained speed-up was approaching the theoretical limit of 4xN, where N is the number of CPU cores used. In case of distributed calculations the networking overhead was not negligible but still acceptable compared to the total computing time.

The use of pure multi-threading or SSE SIMD instructions reaches the theoretical speed-up limits (N or 4, respectively), however these speed-up values are far below the ones of the GP-GPU or combination of SSE and multi-threading mentioned in previous paragraphs.

When the multi-threading is considered, the best utilisation of the multiple CPU cores (in terms of CPU utilisation and load balancing) can be obtained by the use of pool of worker threads, each of them 'pinned' to a single CPU core and by splitting the calculation to a stream of smaller tasks. The

Operation	***non-cached***		***cached***	
	original / *s*	***optimised*** / *s*	***original*** / *s*	***optimised*** / *s*
TIFF load	2.65	2.73	0.86	0.59
Region Growing Segmentation	0.94	0.98	0.94	0.99
Segm. Clips, Areas and Perimeters	1.15	1.00	1.15	1.00
Segments' Geometric Properties	2.04	0.18	1.99	0.18
DATA save	1.00	0.37	1.01	0.31
TIFF save	1.20	0.78	1.15	0.71
TOTAL	**8.99**	**6.04**	**7.10**	**3.85**
time spent by I/O operations	54%	64%	43%	56%

Table 2: MEEO segmentation: comparison of the execution times of the original MEEO and optimised ISS implementation for a sample labelled image (7661x4667 pixels, 34.2 MiB). The execution times for non-cached and cached input data are compared as well. The displayed values are average values of twelve independent measurements. HW configuration: AMD Phenom II X4 @ 3.0 GHz, 4 GiB DDR3 @ 1.33 Ghz, single 512 GB SATA II.

trivial fork-join pattern of computing (splitting work to N task for N CPU cores), although in specific cases may be satisfactory, generally performs worse due to the inherent lack of load balancing.

It is worth to mentioned, the iterative K-means means algorithm is an example of CPU bound processing and therefore it has large potential for performance improvement. On contrary, the performance of I/O bound algorithms (such as many simple single pass image operations) may be limited by the disk storage I/O rates.

The results for different versions of the K-means algorithm are summarised in Tab. 1.

2.2 Algorithms' Implementation

In this section we briefly summarise the implementation of performance optimised variants of the algorithms selected in agreement with ESA [AD06-AD11].

These algorithms were: k-means (already covered by the previous section), OTB pan-sharpening, MEEO segmentation, and OTB Road Extraction algorithm.

1.1.1 MEEO segmentation

The MEEO segmentation algorithm performs segmentation of labelled images (i.e., after the clustering has been performed and an integer labels have been given to similar pixels belonging to the same clusters). The algorithm identifies isolated regions of pixels (segments) and calculates the segments' geometrical properties (perimeter, area, rectangularity, etc.).

Since this single pass algorithm is significantly bound by the I/O rate we have concluded that application of radical calculation performance improvement would be meaningless, as the I/O transfers would always dominate. Nonetheless careful utilisation of muti-threading helped to reduce the computing time to approximately half of the original value (Tab. 2). The optimisation

time	**original OTB** algorithm	**LPFPanSharp** single thread	**LPFPanSharp** multi-thread	**LSQPanSharp** single thread	**LSQPanSharp** multi-thread
non-cached	19 min 19 s	2 min 5 s	2 min 35 s	2 min 59 s	2 min 20 s
cached	18 min 45 s	47 s	24 s	1 min 1 s	35 s

*Table 3: Pan-Sharpening: comparison of the execution times of the original OTB Simple Sharpening implementation, the same reimplemented algorithm (both single and multi-thread variants; **LPFPanSharp**), and the least-square based pan-sharpening algorithm implementation (both single and multi-thread variants; **LSQPanSharp**). The execution times for non-cached and cached input data are compared as well. Low pass filter: BoxCar 7x7 pixels. Processed data: GeoEye-1 (free sample); panchromatic image 13532x31624 pixels, multi-band image 3383x7906x4 pixels (prior test re-sampled to dimension of the panchromatic image). The displayed values are average values of twelve independent measurements. HW configuration: AMD Phenom II X4 @ 3.0 GHz, 16 GiB DDR3 @ 1.33 Ghz, 2x 256 GB SATA II SW-RAID0.*

included the improvement of the output image data saving due to better image block manipulation and muti-threaded generation of the plain-text output.

1.1.2 Orfeo-Toolbox algorithms

As we, during the course of the project, had to deal with the Orfeo-Toolbox, we have found out that reasons of very poor performance of the toolbox algorithms are typically [RD02]:

- wrong memory management leading to unreasonably high allocation of RAM which leads to significant performance degradation due to the memory swapping (e.g., road extraction from ~200 MiB AVNIR-2 product may require ~ 9 GiB of RAM).
- badly written specific application filters (e.g., minor modification of the OTB LinkPath filter reduces may execution time from almost 2 hours to less than 2 minutes).

1.1.3 OTB pan-sharpening algorithm

The pan-sharpening, in-general, takes a panchromatic image (having better spatial resolution) and a spectral multi-band image (lower spatial resolution and co-registered to panchromatic image's geometry) and enhances the multi-band images by finer image features of the panchromatic images.

Specifically in case of the of the Simple OTB panchromatic algorithm, a simple low pas (LP) filter (2D Box-Car) is applied to the pan-chromatic image. The pixels original pan-chromatic image is divided by the LP filtered version of the same image. This pixel ratios are than multiplied by pixel values of each band leading to the final pan-sharpened image.

After the analysis of the OTB implementation we have found that the performance is acceptable for smaller images provided that the OTB memory footprint fits into available RAM. For very large images however the performance becomes suboptimal (Tab. 3).

We have reimplemented the algorithm using processing by image tiles reducing the memory footprint. We did not apply any kind of radical performance optimisation except the multi-threading as the simple pan-sharpening is strongly I/O bound operation.

In addition we have implemented modification of the OTB simple algorithm replacing the LP filtered image by the linear combination of the bands of the source multi-band image. The parameters of

FILE ID	original OTB time / hour:min:sec	ISS LinkPath time / hour:min:sec	Speed-Up
1-3	01:58:03	00:01:14	95.2
1-14	00:09:19	00:00:40	14.0

Table 4: OTB Road-Extraction from AVNIR-2 products – the best and worst cases and improvement after replacement by the optimised LinkPath filter. HW configuration: AMD Phenom II X4 @ 3.0 GHz, 16 GiB DDR3 @ 1.33 Ghz, 2x 256 GB SATA II SW-RAID0. For more details see [RD02].

the linear combination are detected automatically by simple least-square fitting.

The benchmark results are presented in Tab. 3.

1.1.4 OTB road extraction

The OTB road extraction algorithm is composed of set OTB filters. At the beginning, the spectral distance from a single reference pixel (road value) value is calculated reducing the input multi-band image to single band one. Afterwards, set of raster processing filters is applied highlighting the linear features of the image with minimal spectral distance from the reference pixel value. In the second part of the algorithm the linear features are converted to vector presentation and the vectorised linear segments are finally filtered and merged to line segments most-likely representing the roads.

During the algorithm analysis we have identified that the major performance bottle-neck is the, so called, *Link-Path* filter responsible for joining of short path fragments to longer paths, where a path is a set of connected line segments defined by a sequence of discrete points (nods).

The Link-Path filter, as implemented by the OTB, calculates the distance between the path pairs which involves calculation of Euclidean distance between the nods of each path fragment. Close fragments are than linked together. The excessive calculation of the Euclidean distance between nodes whose path are even very distant is responsible for the very long computing times for the images of urban areas where the density of the roads is high.

We have replaced the original OTB LinkPath filter by a new one utilising following improvements:

– By using the bounding boxes (inflated by the distance threshold) we can quickly reject comparison of paths having non-overlapping bounding box without calculations of Euclidean distances between all their nods.

– Once the bounding boxes are used, it is very simple to split the whole are to smaller

FILE ID	single thread / CPU time / sec	multi-thread / CPU time / sec	CUDA / GPU time / sec	CUDA / GPU 2x time / sec
1-3	23.45	9.72	11.95	12.43
1-14	23.23	9.20	12.96	13.02

Table 5: Execution times of the re-implemented raster processing part of the OTB Road-Extraction algorithm for the same AVNIR-2 products as in Tab. 4. The results for reference single thread, multi-threaded, one and two GP-GPU cards are compared. HW configuration: AMD Phenom II X4 @ 3.0 GHz, 8 GiB DDR3 @ 1.33 Ghz, 2x NVidia GTX 285 2 GiB DDR2, 2x 256 GB SATA II SW-RAID0. For more details see [RD02].

(partially) overlapped subsets. That way we reduce the number of compared paths.

By this improvements, we have managed to speed-up the OTB road extraction by from ~10 (rural areas) to ~100 times (urban areas). Further the computing time have been stabilised so that the computing time is practically independent of the road density of the scene (Tab. 4).

We have also reimplemented the raster processing part of the OTB algorithm from scratch reducing significantly the enormous RAM demands of the OTB algorithm. The multi-threaded version and CUDA (GP-GPU) version. As this algorithm is significantly I/O bound (the computing time is comparable to time needed by reading input and output data) the results do not exhibit large speed-ups (Tab. 5).