PARALLELISATION OF THE MODEL-BASED ITERATIVE RECONSTRUCTION ALGORITHM DIRA

Alexander Örtenberg, Maria Magnusson, Michael Sandborg,
Gudrun Alm Carlsson and Alexandr Malusek

Original Publication:

Alexander Örtenberg, Maria Magnusson, Michael Sandborg, Gudrun Alm Carlsson and Alexandr Malusek, PARALLELISATION OF THE MODEL-BASED ITERATIVE RECONSTRUCTION ALGORITHM DIRA, 2015, Radiation Protection Dosimetry.
http://dx.doi.org/10.1093/rpd/ncv430

Parallelization of the model-based iterative reconstruction algorithm DIRA

A. Örtenberg[1], M. Magnusson[1,2], M. Sandborg[1], G. Alm Carlsson[1] and A. Malusek[1]

[1]Medical Radiation Physics, Department of Medical and Health Sciences and Center for Medical Image Science and Visualisation, Linköping University, SE-58185 Linköping, Sweden

[2]Computer Vision Laboratory, Department of Electrical Engineering, Linköping University, SE-58183 Linköping, Sweden

Corresponding author: alexandr.malusek@liu.se

fax +46 101032895

phone +46 101033059

Short running title (max 40 characters): Parallelization of DIRA

Parallelization of the model-based iterative reconstruction algorithm DIRA

A. Örtenberg, M. Magnusson, M. Sandborg, G. Alm Carlsson and A. Malusek

ABSTRACT

New paradigms for parallel programming have been devised to simplify software development on multi-core processors and many-core graphical processing units (GPU). Despite their obvious benefits, the parallelization of existing computer programs is not an easy task. In this work, the usage of the Open Multiprocessing (OpenMP) and Open Computing Language (OpenCL) frameworks is considered for the parallelization of the model-based iterative reconstruction algorithm DIRA with the aim to significantly shorten the code's execution time. Selected routines were parallelized using OpenMP and OpenCL libraries; some routines were converted from MATLAB to C and optimized. Parallelization of the code with the OpenMP was easy and resulted in an overall speedup of 15 on a 16-core computer. Parallelization with OpenCL was more difficult owing to differences between the CPU and GPU architectures. The resulting speedup was substantially lower than the theoretical peak performance of the GPU; the cause was explained.

INTRODUCTION

Technological improvements in computer hardware such as multi-core processors and many-core graphical processing units may speed up computations to such an extent that codes originally usable only in research environments achieve execution times acceptable in clinics. The new architectures bring problems that were non-existent in the single processor architectures. To aid the developers paradigms for parallel programming have evolved that provide easy to use methods by abstracting the tedious tasks of low-level thread and core management. In this work the Open Multiprocessing (OpenMP) and Open Computing Language (OpenCL) paradigms are considered.

OpenMP[(1)] is a specification for a set of compiler directives, library routines, and environment variables that can be used to specify high-level parallelism in FORTRAN and C/C++ programs. It is supported by a wide range of compilers, such as the GNU Compiler Collection (GCC) and Intel C++ Compiler (ICC), and most processor architectures and operating systems. It uses a set of preprocessor directives and keywords to generate multithreaded code at compile time for a shared memory system.

OpenCL[(2)] is an open, royalty-free standard for cross-platform, parallel programming of modern processors found in personal computers, servers and handheld/embedded devices, which allows for execution on a multitude of different hardware architectures. OpenCL executes on devices consisting of one or more compute units, which in turn are divided into processing elements performing the computations. The programmer must specify a number of work-items to use. OpenCL automatically assigns each work-item an identification number (global id) and each work-item executes an instance of the kernel (the code). The work-items are mapped onto the processing elements of the device.

DIRA[(3)] is a MATLAB implementation of a model based iterative image reconstruction algorithm in dual-energy computed tomography (DECT). It can be used for the determination of elemental

composition of tissues in radiation treatment planning. DIRA employs automatic segmentation of tissues. The segmented tissues are mathematically decomposed into pre-defined material bases from maps of linear attenuation coefficients obtained at two different x-ray tube voltages. In case of a two-material decomposition method (MD2), mass density of the imaged tissue and mass fractions of doublet components are determined for each voxel. In case of a three-material decomposition method (MD3), an extra assumption is made about the mass density of the imaged tissue and mass fractions of triplet components are determined. Elemental composition of the imaged tissue is then determined from known elemental compositions of base material doublets or triplets.

As the problem does not have a unique solution, DIRA is used as a test bed for the development and testing of new approaches. It has to be flexible and easy to use, but it also has to be fast enough so that optimization tasks requiring reconstruction of many slices can be done in a reasonably short time. A conversion of the whole source code from MATLAB to C would speed up DIRA, but it would also make the code difficult to read for new developers. A decision was made to convert and parallelize routines where a notable impact on performance was expected. However, there was no general recipe on how to parallelize codes like DIRA for the new computer architectures.

The aim of this work is to demonstrate how code parallelization via the OpenMP and OpenCL libraries can be done and what speedup can be achieved in the case of iterative image reconstruction algorithms. The model-based iterative reconstruction algorithm in dual-energy computed tomography DIRA is used as an example, but the results may be applicable to other iterative image reconstruction algorithms too.

MATERIALS AND METHODS

Data processing in DIRA

DIRA converts measured polyenergetic projections to “ideal” monoenergetic ones by applying a calculated correction. The monoenergetic projections are then reconstructed using the filtered back-

projection method. The reconstructed data are used for the construction of a physical model of the imaged object. The physical model is used to calculate the projection correction. Data-flowchart of DIRA is in figure 1.

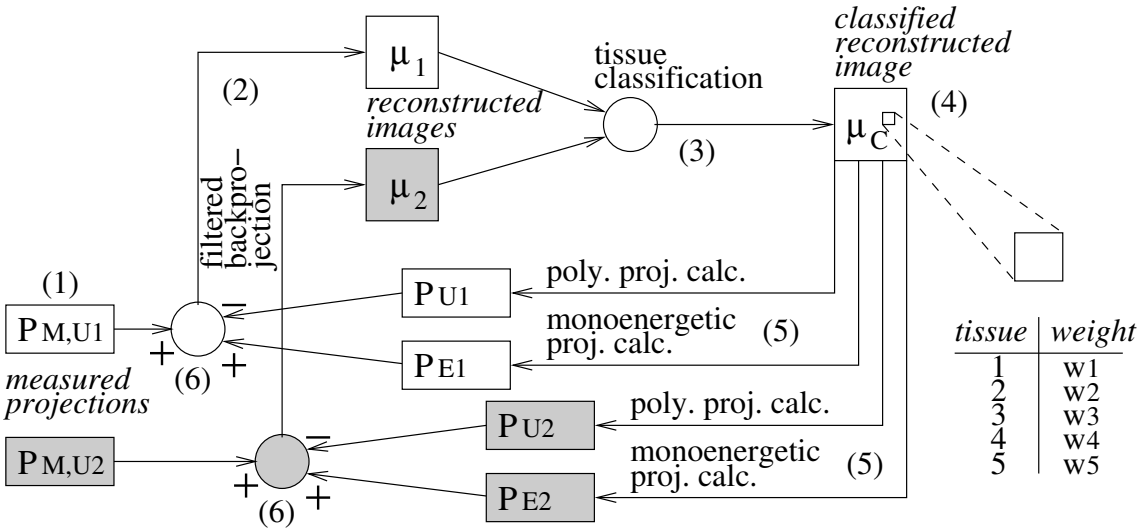

Figure 1. Data-flowchart of the model-based iterative reconstruction algorithm DIRA. See the text for explanation.

Data processing is done in the following steps:

1. Two sets of measured projections $P_{M,U1}$ and $P_{M,U2}$ taken at two different tube voltages U_1 and U_2, respectively, are used as input.
2. $P_{M,U1}$ and $P_{M,U2}$ are reconstructed via filtered backprojection into the volumes μ_1 and μ_2 containing the linear attenuation coefficients at photon energies approximately equal to the effective energies E_1 and E_2 of the spectra for the tube voltages U_1 and U_2.
3. The reconstructed volumes μ_1 and μ_2 are classified to preselected tissues, e.g. bones and soft tissue in this example.

4. The soft tissue is then decomposed using three-material decomposition (MD3) and the bone is decomposed using two-material plus density decomposition (MD2)[4], resulting in the decomposed volume μ_C.
5. The monoenergetic projections P_{E1} and P_{E2} at the energies E_1 and E_2, and the polyenergetic projections P_{U1} and P_{U2} for spectra U_1 and U_2 are forward projected using Joseph's method[5].
6. The differences between the measured polyenergetic projections $P_{M,U1}$ and $P_{M,U2}$ and the calculated polyenergetic projections P_{U1} and P_{U2} are first computed and then added to the monoenergetic projections P_{E1} and P_{E2}. The results are sent to the next iteration starting with filtered backprojection.
7. After a number of iterations, $P_{M,U1}$ and $P_{M,U2}$ almost perfectly match P_{U1} and P_{U2} and give no contribution to the reconstructed images, the beam hardening artefacts are removed and accurate mass fractions of the base materials as well as the density for the bone tissue obtained.

The material classification is done as follows. A segmentation method (for instance a threshold segmentation) determines the kind of tissue the voxel belongs to; a list of possible tissues is defined by the user. Soft tissues are typically decomposed to a material triplet, for instance lipid, protein, and water using the MD3 method. Bone tissues are decomposed to a material doublet, for instance compact bone and bone marrow using the MD2 method. Each segmented tissue may use its own material doublet or triplet.

Changes to the code

The DIRA code was optimized and selected routines were parallelized using OpenMP and OpenCL libraries. The optimization was done by converting the MATLAB code to C and changing the

forward projection routine from destination-driven to source-driven, see Beister et al.[6] for more information about the latter two approaches.

Forward projections (step 5 in previous section) are calculated using the *sinogramJc* and *computePolyProj* functions. The former computes projections (sinograms) for each base material individually. The latter then uses these material-specific projections to calculate polychromatic projections, which simulate the measured projections. The old implementation of the projection generation used a destination-driven method. The drawback of this method is that calculating the line integrals is expensive. The calculation of the sum of interpolated values requires stepping through the image of mass fractions, where each step has to be checked against the boundaries of the image. These checks are expensive as they introduce one branching operation for each check. The new implementation of the projection generation uses a source-driven method. Each individual pixel's contribution to each projection is calculated. Pixels outside of a circle with its centre in the middle of the image of mass fractions can produce coordinate values outside of the sinogram; these are excluded. Also, a pixel in the input image must have a non-zero value to contribute to the projections. The number of pixels to process depends on the input image. As an example, excluding half of the pixels in the mass fraction image reduces the computational load by half and the execution time is halved.

The filtered backprojection (step 2 in previous section) was originally done using the built-in MATLAB *iradon* function, which is written in C and parallelized. Mathworks has not released details about the parallelization. A new implementation based on the work by Orchard [7] was made (function *inverseRadon* and associated functions). The general implementation of backprojection is to place each value of a projection along a line through the output image. This implementation suffers from low performance as the placement of values in the output image is performed out of order, which causes frequent cache misses and slow memory accesses. The new implementation

changes the order in which the projection values are placed onto the output image. For each projection the projection values are placed onto each row in the output image in the correct position.

Performance evaluation

Performance of the code was evaluated on (i) a notebook (dual-core Intel T1600 CPU), (ii) a Z400 workstation with a GPU (quad-core Intel W3520 CPU, AMD R7 260X GPU), (iii) a workstation (quad-core Intel 3570K CPU), and (iv) one node of the supercomputer Triolith (two Intel E5-2660 CPUs, 16 cores per node) at the National Supercomputer Centre at Linköping University, Sweden for reconstruction of a mathematical anthropomorphic phantom (figure 2). The phantom is defined in the 'slice113' example in the cmiv-dira code repository on GitHub[8]. In this example, 560 projections $P_{M,U1}$ and $P_{M,U2}$ (step 1) were simulated using the Drasim code[9] for 512 detector elements in an axial scanning geometry. Predefined masks simulating an ideal segmentation separated tissues to bone and soft tissue. The material doublet for bone consisted of compact bone and bone marrow, the material triplet for soft tissue consisted of lipid, protein, and water. Dimensions of reconstructed images were 511×511; the number of iterations in DIRA was 4.

The speedup factor of the parallelized code was evaluated as $S = T_s/T_p$, where T_s and T_p are the execution times of the sequential and parallel implementation, respectively.

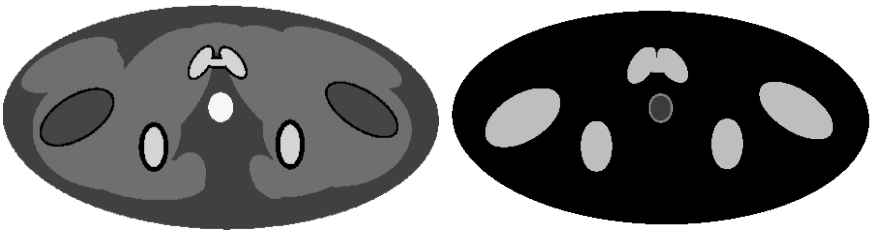

Figure 2. Left: The mathematical anthropomorphic phantom used in the slice 113 example. Grey levels show the adipose tissue, muscle, compact bone, bone marrow and calcified prostate. Right: The mask used to separate soft tissues, bones and prostate.

RESULTS

Figure 3 shows that speedup factors of 2 - 4 were achieved by optimizing the code: changing selected routines from MATLAB to C code and from destination-driven to source-driven algorithms. Further speedup was achieved by parallelization using OpenMP.

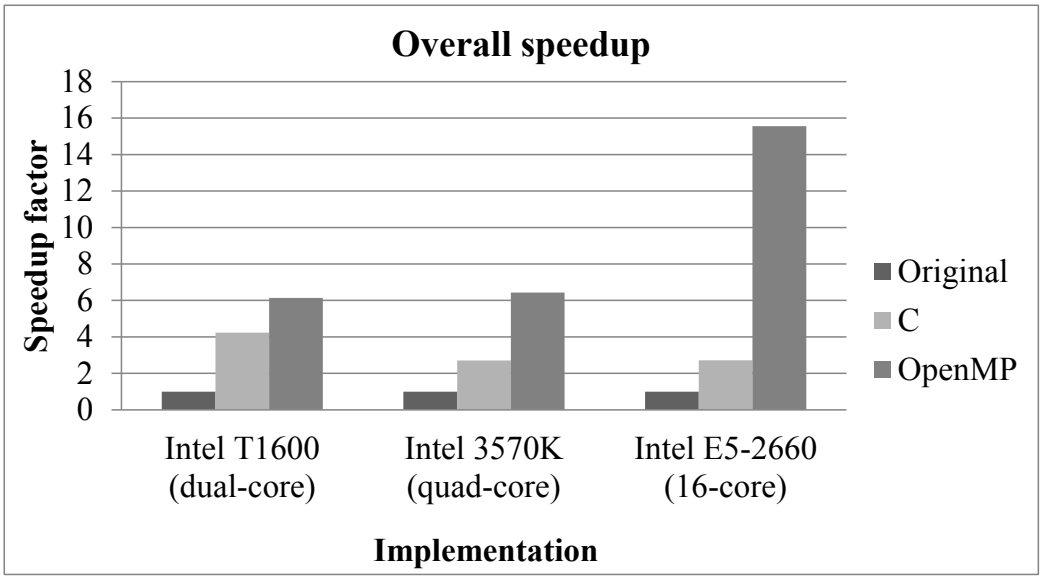

Figure 3. The speedup factor for the original implementation (combination of C and MATLAB code), the optimized implementation in C and the parallel implementation using OpenMP on the 2-core, 4-core and 16-core computers.

Figure 4 shows that the speedup factor increased with increasing number of CPU cores on the 16-core computer. The ideal speedup factor for the 16-core system is 16. The implementation reached a speedup factor of about 10.

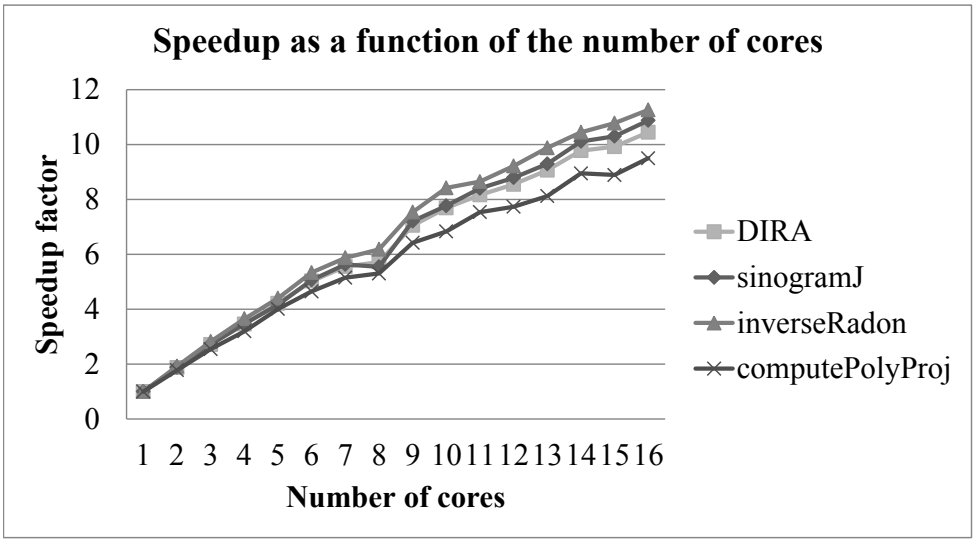

Figure 4. Speedup factor as a function of the number of cores on the 16-core computer with two Intel Xeon E5-2660 CPUs.

Table 1 shows good performance of the OpenMP code on both the 16-core and 4-core computers. In case of sinogramJ, the execution time on the 4-core computer was even 5 times shorter for the OpenMP version. The performance of the GPU code was better than the performance of the single-threaded code, but it was very far from the theoretical peak performance of the GPU and not notably different from the performance of the OpenMP implementation on the 4-core computer. An analysis showed that the poor performance was caused by frequent calls of short, data-demanding routines.

Compared to the original MATLAB version, an overall speedup factor of 15 was achieved on the 16-core system; the execution time of DIRA was reduced from about 2 minutes to just 8 seconds.

Table 1: Execution times (in seconds) of the *sinogramJ*, *computePolyProj* and *inverseRadon* DIRA functions at the 16-core (Triolith) and 4-core (Z400) computers for the C, OpenMP and OpenCL implementations.

	Triolith		Z400		
	C	OpenMP	C	OpenMP	OpenCL
sinogramJ	10.59	0.97	11.16	2.22	3.25
computePolyProj	11.25	1.18	11.09	3.20	1.87
inverseRadon	22.11	1.96	9.51	4.03	4.29

DISCUSSION

A project whose aim is to parallelize a scientific code must address two issues: (i) a suitable software framework must be selected and (ii) the scale of modifications of the code must be decided. In this work, we opted for the freely available OpenMP and OpenCL frameworks. Alternatives to OpenMP and OpenCL are the Message Passing Interface[10] (MPI) for CPUs and the Compute Unified Device Architecture[11] (CUDA) for GPUs, respectively. In case of CPUs, we preferred OpenMP to MPI as the utilization of a distributed memory system was not required. In case of GPU, we preferred OpenCL to CUDA since the latter can only be used on NVIDIA's GPUs. OpenCL on the other hand is a multi-platform solution.

The DIRA algorithm is under active development. To make the work of programmers easier, an easy to read code is preferred. For this reason, we opted for the parallelisation of critical routines only and avoided any radical redesign of the code. The drawback was that speedup factor was rather low for the GPU, where the DIRA's approach did not scale well. An interesting question is whether there is an approach applicable in DIRA that is capable of notably better utilization of the GPU.

DIRA's approach to image reconstruction has been unique for a long time and so there is little possibility to compare the parallelization of DIRA with the parallelization of other projects. Some information is available about parallelization of classic image reconstruction algorithms like the filtered backprojection or noise-suppressing iterative image algorithms[6]. In general, however, vendors do not disclose implementation details of their algorithms. Some quantitative data are available for MCGPU, which is a massively multi-threaded GPU-accelerated x-ray transport simulation code[12]. It can generate radiographic projection images and CT scans of the human anatomy. It uses the CUDA library for NVIDIA GPUs and the MPI library for multiple GPUs. MCGPU achieved a speedup factor of 6 on a NVIDIA C2070 GPU compared to a quad-core Xeon processor[13]. This code, however, produces forward projections only. It neither reconstructs CT images nor performs material decomposition.

Beister et al.[6] describe an implementation of an iterative reconstruction algorithm using CUDA for a GPU. The authors used the ordered subset convex algorithm, which updates image estimates in multiple steps using a subset of all projections. For the forward projection they used a destination-driven method by multi-sampling x-rays through the voxels and averaging the results. Each thread on the GPU was assigned a single detector pixel. The authors argued that the destination-driven approach is better suited for the GPU than the source-driven approach. In the case of DIRA, however, no notable benefit of the destination-driven approach was observed. In fact, this approach on the AMD R7 260X GPU did not show any notable speedup compared to the source-driven approach on the quad-core Intel W3520 CPU[14].

CONCLUSIONS

The code was parallelized using both the OpenMP and OpenCL frameworks. For multi-core CPUs, parallelization of selected routines with OpenMP was easy, required only small changes to the code, and scaled almost linearly with the number of CPU cores. For many-core GPUs, parallelization with

OpenCL was more difficult owing to differences between the CPU and GPU architectures. Moreover the achieved speedup was far from the theoretical one for the parallelization of short, data-demanding routines. Of interest is that a notable speedup was achieved by simple code optimization: a conversion from MATLAB to C, and a transformation from a destination-driven to a source-driven method in forward projection generation.

FUNDING

This work was supported by the Swedish Cancer Foundation [grant numbers CAN 2012/764, CAN 2014/691].

ACKNOWLEDGEMENTS

This work has been conducted within the Centre for Medical Image Science and Visualization (CMIV) at Linköping University, Sweden. The authors thank the National Supercomputer Center at Linköping University for providing access to its hardware; namely Peter Kjellström and Peter Münger.

REFERENCES

1. OpenMP ARB Corporation. Frequently asked questions on OpenMP. Available at: http://openmp.org/openmp-faq.html#WhatIs (26 January 2015, date last accessed)
2. Khronos Group. The open standard for parallel programming of heterogeneous systems. Available at https://www.khronos.org/opencl/ (26 January 2015, date last accessed)
3. Magnusson, M., Malusek, A., Muhammad, A. and Alm Carlsson, G. Iterative reconstruction for quantitative tissue decomposition in dual-energy CT. In: Heyden, A. and Kahl, F. (Eds.) Image analysis. Springer Berlin Heidelberg, Berlin, Heidelberg, 479–488 (2011).

4. Malusek, A., Karlsson, M., Magnusson, M. and Alm Carlsson, G. The potential of dual-energy computed tomography for quantitative decomposition of soft tissues to water, protein and lipid in brachytherapy. Phys. Med. Biol. 58, 771-785 (2013).
5. Joseph, P.M. An improved algorithm for reprojecting rays through pixel images. IEEE Trans. Med. Imaging, Vols. MI-1, no. 3, 192-196 (1982).
6. Beister, M., Kolditz, D. and Kalender, W.A. Iterative reconstruction methods in X-ray CT. Physica Medica 28, 94–108 (2012).
7. Orchard, J. MATLAB Central File Exchange: iradon_speedy. Available at: http://www.mathworks.com/matlabcentral/fileexchange/12852-iradon-speedy (19 November 2014, date last accessed)
8. Malusek, A. GitHub: cmiv-dira. Available at: https://github.com/AlexandrMalusek/cmiv-dira/ (13 February 2015, date last accessed)
9. Stierstorfer, K. DRASIM: a CT-simulation tool. Internal report - Siemens Medical Engineering (2007).
10. T. O. M. Project. Open MPI: open source high performance computing. Available at: http://www.open-mpi.org/ (26 January 2015, date last accessed)
11. NVIDIA. What is CUDA? Available at: http://www.nvidia.com/object/cuda_home_new.html (26 January 2015, date last accessed)
12. U. S. Food and Drug Administration. mcgpu - Monte Carlo simulation of x-ray transport in a GPU with CUDA. Available at: https://code.google.com/p/mcgpu/ (26 January 2015, date last accessed)
13. Tisseur, D., Andrieux, A., Costin, M. and Vabre A. Monte Carlo simulation of the scattered beam in radiography non-destructive testing context. In joint international conference on supercomputing in nuclear applications and Monte Carlo 2013, Paris, France, (2013).

14. Örtenberg, A. Parallelization of DIRA and CTmod using OpenMP and OpenCL. Master's thesis, Linköping University (2015). Available at: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-119183