Order of magnitude speedup for iterative algorithms in quantitative susceptibility mapping using multi-core parallel programming

D. Cui[1,2], T. Liu[1], P. Spincemaille[3], and Y. Wang[4]

[1]Biomedical Engineering, Cornell University, New York, NY, United States, [2]Optic electronics, Beijing Institute of Technology, Beijing, China, People's Republic of,

[3]Radiology, Weill Cornell Medical College, New York, NY, United States, [4]Weill Cornell Medical College

Introduction

Calculating the susceptibility distribution responsible for a measured magnetic field is a well known ill-posed inverse problem [1]. Different methods have been proposed to overcome the ill-posedness, either through data acquisition strategy [2] or through regularization [3, 4]. These algorithms solve the problem iteratively by repeatedly applying the forward problem and updating the solution based on residue. In each iteration, dipole convolutions are done in the Fourier domain by performing point-wise multiplications while phase noise whitening is done in the image domain. The four Fast Fourier Transforms (FFT) needed for each iteration constitute a major bottleneck preventing fast image reconstruction. In this study, a parallelized FFT based on OpenMP was implemented to achieve quasi real-time susceptibility map reconstructions.

Methods

Weighted L2 gradient regularization (wL2) computes the susceptibility distribution χ by minimizing the following penalty function: $\chi = \arg\min \sum \left\| w(r) \times (\delta_B(r) - d \otimes \chi) \right\|_2^2 + \alpha \left\| w_G(r) \times G\chi \right\|_2^2$. This problem has a quadratic form and various numerical methods may be employed to solve the problem iteratively. A conjugate gradient algorithm repeatedly calculates the forward problem and uses the residual to update the solution. The minimization problem can be equivalently written as $FT^{-1}\left(D \cdot FT\left(\cdot w^H \cdot wFT^{-1}\left(D \cdot FT\left(\chi_p\right)\right)\right)\right) + \alpha\left(G^H w_G{}^H w_G G\chi\right) = FT^{-1}\left(D \cdot FT\left(w^H w \cdot \delta_B\right)\right)$. Here four Fourier Transforms are used in every iteration and typically, hundreds of iterations are required for convergence.

MPI (Message Passing Interface), OpenMP (Open Multi-Processing) and CUDA (Compute Unified Device Architecture) are three well-known programming interfaces for parallel computing. MPI is often used as the distributing interface on the cluster servers when the calculation can be efficiently divided into smaller independent problems. The OpenMP is an application programming interface (API) that supports multi-platform shared memory multiprocessing programming in C, C++ and Fortran on many architectures, and platforms. It has better performance for those problems which have a large amount of data sharing and switching on a multi-core CPU. CUDA is a GPU computing interface and it depends on nVidia special graphics cards to provide a powerful parallel computing environment. However, CUDA cannot work on non-support graphics card such as ATI graphics card. The weighted L2 gradient regularization is an iterative reconstruction algorithm that is hard to disperse. However, the Fourier transform can be computed in parallel and is ideally suited for OpenMP parallel optimization.

The reconstruction processing of the 3D data may be dispersed into X, Y, and Z directions such that the reconstruction processing is parallelizable along the spatial dimensions. We designed a parallel optimization reconstruction algorithm, which processes several data blocks that are shared in the memory. The flow chart is shown in Figure 1. In iteration step, every block will be processed in parallel in different threads. To maximize usage of the CPU resources, we enable the 8 threads on a quad core CPU with the Hyper-Threading technology. Thus, the program is designed for up to 8 threads. Experimental data consisted of a 240 × 180 × 90 32bit float point matrix. Iterative precision was set to 0.1%. As the purpose of this paper is to accelerate the wL2 for real-time application, we implemented the parallel wL2 in C/C++ and compared with other implementations: a) Matlab 2007b non-multithreaded, b) Matlab 2009a non-multithreaded c) Matlab 2009a multithreaded d) FFTW 3.2 non-multithreaded, e) Parallel C/C++ with 4 threads, and f) Parallel C/C++ with 8 threads. Calculation were done on a Dell Studio XPS 435MT, Intel Quad core i7 920 CPU 2.66GHz, Vista SP2 64bit OS. Intel C++ Compiler 11.1 is used for FFTW C code and OpenMP parallel C/C++ code compiling. Matlab Compiler 4.6 (2007a)/ 4.10 (2009a) with C/C++ math library is used for M code compiling.

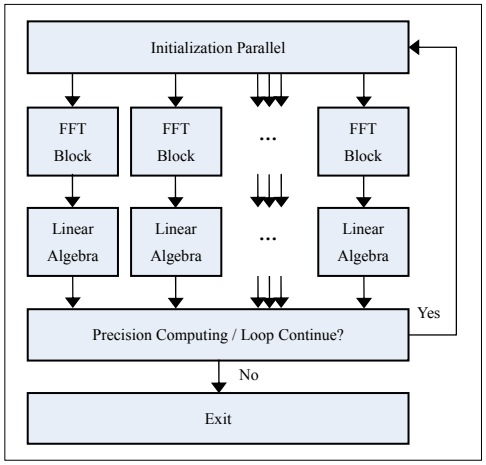

Figure 1 Flow Chart of Parallel Optimization

	a	b	c	d	e	f
Usage of CPU/%	13	14	32	13	50	92
Time/s	769.7	621.5	407.6	301.1	60.1	38.4
Speedup	1.00	1.24	1.89	2.56	12.81	20.04

Table 1 Reconstruction Time Comparison

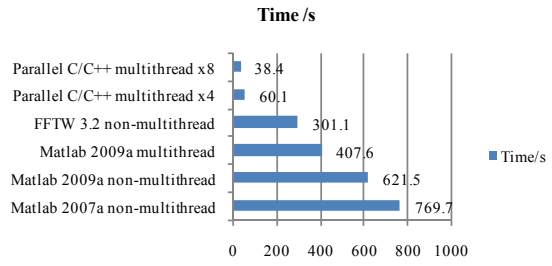

Figure 2 Reconstruction Times

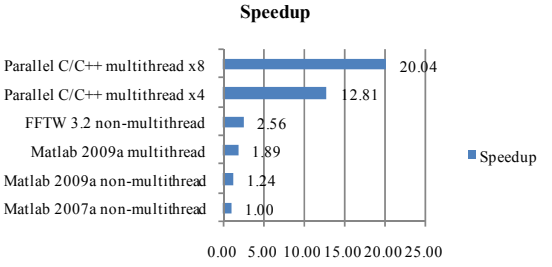

Figure 3 Reconstruction Speedup

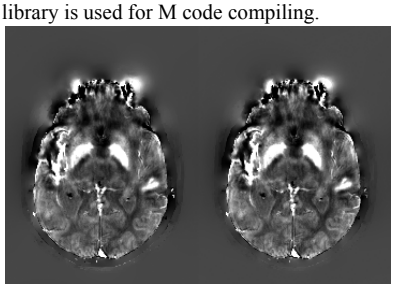

Figure 4. Reconstruction Images

Result

Reconstruction times and speedup are shown in Table 1, Figure 2 and Figure 3; Parallel C/C++ with 8 threads wL2 can maximize usage of the quad core CPU resources in virtual 8 cores mode. In addition, it is 10 times faster than the fastest optimized Matlab program.

Reconstructed images are show in Figure 4, with a parallel wL2 reconstruction image on the left and a Matlab wL2 reconstruction image on the right.

Discussion and conclusion

A considerable computational speed up was achieved using multithreaded parallel processing algorithm to accelerate the FFT bottleneck in the iterative conjugate gradient solver. The code was an order of magnitude faster that the fastest optimized matlab code. The iterative conjugate gradient algorithm for the reconstruction of the susceptibility maps can be further accelerated by using a higher end workstation and it is estimated that it can be achieved in 10 second using a 32 core server for near real-time application with the ultimate goal of bringing quantitative susceptibility mapping to clinical practice.

Ref: [1] Haacke et al, MRI:23:1-25; [2] Liu et al, MRM:61:196-204; [3] de Rochefort et al, MRM: *in press;* [4] Kressler et al, IEEE TMI:2009.