Scientific Programming 10 (2002) 253–261
IOS Press

OpenMP programming for a global inverse model

Ping Wang[a,*] and Xiaoping Wu[b]
[a]*Lawrence Livermore National Laboratory, PO Box 808, L-312, Livermore, CA 94551, USA*
Tel.: +1 925 423 2612; Fax: +1 925 422 3134; E-mail: Wang32@llnl.gov
[b]*Jet Propulsion Laboratory, CA, USA*

Abstract. The objective of our investigation is to establish robust inverse algorithms to convert GRACE gravity and ICESat altimetry mission data into global current and past surface mass variations. To assess separation of global sources of change and to evaluate spatio-temporal resolution and accuracy statistically from full posterior covariance matrices, a high performance version of a global simultaneous grid inverse algorithm is essential. One means to accomplish this is to implement a general, well-optimized, parallel global model on massively parallel supercomputers. In our present work, an efficient parallel version of a global inverse program has been implemented on the Origin 2000 using the OpenMP programming model. In this paper, porting a sequential global code to a shared-memory computing system is discussed; several efficient strategies to optimize the code are reported; well-optimized scientific libraries are used; detailed parallel implementation of the global model is reported; performance data of the code are analyzed. Scaling performance on a shared-memory system is also discussed. The parallel version software gives good speedup and dramatically reduces total data processing time.

Keywords: OpenMP, inverse model, least squares, GRACE, gravity

1. Introduction

Recently, parallel computing systems, strongly driven by many applications which require fast processors and large memory, have been invented with new computer architecture design to handle large scale scientific applications, data processing, and other fields. Because of the new design in parallel computing systems, software developers and users face many new challenges, such as designing new parallel software or using existing codes on modern computers.

Performance of most existing codes on modern computers is seriously constrained because the traditional programming method for a single CPU system will not fully benefit from present supercomputers and parallel systems. These systems require modern programming methods to use their fast CPU and large memory systems, but to develop complete new parallel application codes takes time, and is expensive. For those codes which have been well-implemented and used by many users, porting them to a modern parallel computer is a natural choice. Currently there are two major parallel computer models: a distributed-memory model and a shared-memory model. Between these two systems, a hybrid model such as a cluster of SMPs is also available. Each of these systems might require a different programming model.

On a distributed-memory computing system, a Message Passing Interface (MPI) library [4] is usually used for intercommunication among different computing processors for applications using domain decomposition techniques, while OpenMP directives [2] or system directives are used on a shared-memory system to parallelize sequential codes. A hybrid programming model, which combines these two programming models together, has been investigated recently by many researchers to improve scalability. These three models have their own advantages and disadvantages, but they all have to deal with similar issues such as par-

*Corresponding author.

ISSN 1058-9244/02/$8.00

allel software portability, software reuse and maintainability, and more importantly, the total time required to transfer an existing code to an executable one on advanced parallel systems. The debate about whether the shared memory or message passing paradigm is the best is bound to continue for a while. However, many people believe that the OpenMP programming model allows the general user to gain a reasonable amount of parallelism for a reasonable amount of effort. It is commonly believed that using MPI you should be able to get better parallel speedup and portability but that it may require more complicated programming from the user. Message passing code can also be used on most shared memory systems, and in many cases will be a good choice. For applications when performance and portability become more important, an MPI model might be a good choice, but for other applications when time becomes critical, an OpenMP model can be applied because of its simplicity which allows software developers or scientists to port their sequential codes in a relatively short time and have moderate performance on shared-memory systems.

In this paper, we will focus our investigation on the OpenMP model due to the time constraint and the available computing resources. Our experience using OpenMP programming for a large scale scientific application on the SGI Origin 2000 is reported. Through the specific scientific application, a global inverse model, problems which most existing codes might have are discussed; efficient approaches to these problems, including optimization and parallelization, are given; performance and results are analyzed and reported.

2. A global inverse model

The Gravity Recovery and Climate Experiment (GRACE) mission, to be launched by NASA in early 2002, will provide a revolutionary tool to measure Earth's gravity. The GRACE mission will accurately map the Earth's gravity field and its variations over 5-years. It will have two identical spacecraft flying about 220 kilometers apart in a polar orbit 500 kilometers above the Earth. Each spacecraft will carry a dual frequency microwave transmitter and receiver to measure satellite-to-satellite range variations to an accuracy level of a few μm. Global Positioning System tracking and accelerometer measurements will also be made. The mission will provide scientists from all over the world with an efficient and cost-effective way to map the Earth's gravity with unprecedented accuracy and spatial resolution. The results from this mission will yield crucial information about the distribution and flow of mass within the Earth and it's surroundings. It is likely that a series of ever-improving gravity mapping missions will follow on to provide the climate-change sensitive gravity data with a continuous and longer time-span.

Satellite gravity measurements are directly sensitive to present-day mass variations in Greenland and Antarctic ice sheets (ice mass imbalance), in the oceans (bottom pressure change), and over the land as soil moisture content and ground water changes. Since the Earth is a viscoelastic body, it is still deforming in response to late-Pleistocene and Holocene deglaciation of vast ancient ice sheets. This phenomenon is known as post-glacial rebound (PGR) which significantly contributes to mass redistribution and gravity change. The multiple sensitivities of the gravity data to these global processes present a challenge to unambiguous determination of each event. Such challenge is further complicated by many unknown factors governing the mass redistribution, such as surface layer density variation, grounding line retreat/advancement of the continental ice sheets, mantle viscosity structure, and the timing-geometry of deglaciation [5], lithospheric thickness, among others.

On the other hand, the sensitivities of the gravity provide us with many opportunities to learn more about the Earth when multi-mission and ground based data can be combined with dynamic models governing the mass redistribution processes. In addition to the GRACE gravity data, several radar and laser satellite altimetry missions are either complete, operational, or in preparation. Particularly, the Geoscience Laser Altimeter System (GLAS) on-board the Ice, Cloud and land Elevation Satellite (ICESat) mission will be launched also in 2002 to provide concurrent 3 to 5 elevation change measurements over most area of continental ice sheets. Also, a very valuable set of ground-based data is the large number of relative-sea-level (RSL) records along coastlines dating back many thousands of years (kyr) ago. The goal of our study is to establish and calibrate a valid inverse algorithm to incorporate the multi-mission and/or multi-type dataset and a viscoelastic load-induced Earth deformation model. The algorithm can be used to study separation of geophysical sources of change, to evaluate spatio-temporal resolution and accuracy in inferred surface mass variations, and to assess contributions of each mission or data type. More importantly, the algorithm will eventually be used to convert the data into the climate-change-sensitive sur-

face mass variables described above (e.g., ice mass imbalances in Greenland and Antarctica and their contributions to sea level rise, last glacial maximum height of the massive ancient Laurentian and Fennoscandian ice sheets).

3. A numerical approach

Our inverse strategy involves discretization of the surface mass changes over global grids and time intervals, least-squares solution using different types of data with very large a priori parameter uncertainties, and the construction of suitable averaging kernels over space and time. The global algorithm can also provide a framework for future non-linear inference on the mantle viscosity profile.

Our linear global inverse algorithm currently incorporates, as data: secular GRACE gravity rates up to spherical harmonic degree and order 90, secular elevation rates over all area of current continental ice sheets, including Greenland and Antarctica. Widely distributed RSL histories will also be implemented to the algorithm. The global surface of the Earth is divided into 10502 grid cells with spatial scales of about 200-km, and realistic geographical boundaries. The estimated parameters include secular present-day oceanic, hydrological, and ice mass variations, late-Pleistocene and Holocene deglaciation, and any ice mass variations in the last few thousand years over Greenland and Antarctica after several other continental ice sheets essentially disappeared. A detailed description of the numerical algorithm used for the present study is reported by Wu et al. [9].

Based on the above algorithm, a sequential global inverse code was implemented. To exploit recent advances in computing hardware and software, we use multi-thread super computing platforms available at Jet Propulsion Laboratory (JPL) and implement the code in a well-optimized parallel manner. In this code, the grid parameterization is particularly suitable for parallel operations in evaluating data forms and measurement of partial derivatives. The linear least-squares inversion with a priori information and large amount of covariance propagations and analyses are also very good candidates for vector-optimization and parallel operations.

4. Optimization strategies

The above global inverse model was initially developed on sequential computing systems. The code performance on modern computers is seriously constrained. In order to use these systems efficiently, most parts of the code require certain modifications. We have applied single PE optimization techniques and other strategies to our global inverse model on the Origin 2000. These strategies include memory optimization, effective use of arithmetic pipelines, and usage of optimized libraries [8]. The remainder of this section discusses these strategies and the corresponding improvement in performance of the code on the Origin 2000.

Memory optimization and arithmetic pipelines. The Origin 2000 at JPL uses the R12000 Processors with a 300 MHz clock and is capable of 600 MF lops peak performance. It has 128 nodes with a total 64 GB main memory. It is a Scalable Shared-Memory Processor (SSMP) system that offers the advantages of both shared-memory systems and distributed computing systems. The Origin architecture is an instantiation of a Cache-Coherent Non-Uniform Memory Access (CC-NUMA) Architecture. There are several ways to achieve good performance on the Origin system. One is through effective use of cache; another is through effective use of pipelined arithmetic. The global inverse code uses many multidimensional arrays and indirect indexing calculations, which can be inefficient when frequent stride-one addressing is encountered. One improvement is to change to explicit lower dimensional addressing and eliminate indirect indexing calculations. This type of change can increase performance, both by simplifying index calculations and by making the code easier for the compiler to optimize. After these modifications, the code performed with a significant speedup.

Another useful strategy is to unroll loop statements. The R12000 is a four-way super-scalar RISC processor. It can fetch and decode four instructions per cycle to be run on its five independent, pipelined execution units. Although a multiplication or addition can be issued every clock period, the result is not ready for several clock periods. The division, square root, and some other operations need more clock periods. Thus, in order to get top performance from FORTRAN code, the user must expose functional unit parallelism to the compiler. Because of these features of the Origin system, the global inverse code has been optimized by unrolling loop statements if this will expose functional

unit parallelism. This strategy is applied by both manually modification of the code and the compiler options. The number of divide and square root operations also has been minimized by using real variables to store the values resulting from these operation and moving the operations out of loop statements when it is independent of the loop index. After these techniques were applied, the code gained a total speedup of 35%. An example using some of the above techniques is given below:

```
Example 1: single node optimization

! The original code
        .
        do 280 i=1,maxcur
 280    Cpp(ikf(m+i,m+i))=sig7**2
        m=maxgrid+2*maxold+nh*maxcur
        +maxcur
        do 290 i=1,3
 290    Cpp(ikf(m+i,m+i))=sigt0**2
        .

! The new code using explicit
addressing and loop unroll
techniques
        .
        cons1=sig7*sig7
        cons2=sigt0*sigt0
        m=maxgrid+2*maxold+nh*maxcur
        do 280 i=1,maxcur
 280    Cpp(m+i,m+i)=cons1
        m1=m+maxcur
        Cpp(m1+1,m1+1)=cons2
        Cpp(m1+2,m1+2)=cons2
        Cpp(m1+3,m1+3)=cons2
        .
```

Using optimized libraries. There are several optimized libraries available on the Origin 2000, such as the LAPACK library [1]. These libraries have been already highly optimized to give the best possible performance if the user applies them properly. In the global inverse code, one of the major time consuming parts is to solve a least-squares problem. When a fine grid size is used, the total processing time for this part increases significantly. Since the LAPACK has several optimal least-squares equation solvers, the original least-squares equation solver was replaced by calling the corresponding LAPACK subroutines. This substitution results in a significant speedup. The LAPACK least-squares equation solver runs about 4.5 times faster than the original one. The following example shows the replacement of the original code by the LAPACK subroutine for solving a least-squares equation:

```
Example 2: using LAPACK

! The original user's least-squares
equation solver
   .
   call bysian_5(ndata,mpara,Cdd)
   .

! The new code using LAPACK
   .
   call DGELSX(......)
   .
   .
   call DTRTRI(......)
   .
   .
   call DTRMM(......)
   .
```

After all these techniques were applied, the entire optimized code gained about a total speedup of 240%, which dramatically reduced the total wallclock time and makes it feasible to run higher resolution applications. Detailed comparison data are given later.

5. An OpenMP programming model

After the optimization of the code, the next step is to choose a suitable parallel programming model so that the code, with moderate modifications, can be run on a parallel system. It should be noted that it is possible to use a compiler to auto-parallelize some codes, but the results from the auto-parallelizer depend on the complexity of the code and the data layout therein. We first tried the parallel FORTRAN 77 compiler with the *pfa* and *O2* options on the Origin 2000, but the results were not as good as we expected because of the complexity of the code. In general, these options work well for codes with simple and clear data structures. After considering the time constraint (GRACE will be launched by NASA in early 2002), the simplicity of OpenMP programming, and more importantly, the code structure, we decided to use OpenMP directives to implement a shared-memory programming model. OpenMP is the emerging industry standard for directive-based shared-memory parallel processing. This permits the applica-

tion designer to distribute work over multiple processors and thereby develop faster codes. It is a directive based parallel programming model that allows incremental parallelization of codes, i.e. individual loops or subroutines may be parallelized. For those codes which have explicit parallel regions or independent work in a loop, using OpenMP directives is straightforward. But for those codes which have complicated data structures, more care needs to be taken. Our global inverse model has 50,000 lines in FORTRAN with hundreds of subroutines. In order to use OpenMP directives, the following approaches have been used.

First, a performance tool and a timer were used to identify bottlenecks of the code. Once this was done, we worked on the most time consuming part of the code and detected all possible regions and loop statements with heavy work load. For the global inverse code, assembling the global matrix part requires a large amount of processing time, especially in the ocean area. The least-squares solver also requires significant precessing time. Besides those two parts, the interpretation of the solutions is time consuming as well. So most attention was focused on these three parts. After the analysis of the entire code, several types of directives were introduced to the code: OpenMP PARALLEL DO, PARALLEL SECTIONS, PARALLEL DO REDUCTION.

The high-level framework of the global inverse model in Fig. 1 illustrates the major code components for both the original model and the new OpenMP model. For many large loop statements which appear in the matrix formation and the solution interpretation parts, OpenMP Parallel Do directives were applied. For example, gravity due to ocean mass variations was computed through a large loop statement (loop through all grid points covering the ocean area) and several subroutines were called inside this loop. The Parallel Do directive specifies that the iterations of the immediately following Do loop must be executed in parallel. Since the solution from each grid point, at that stage, is independent from each other, an OpenMP Parallel Do directive can be inserted before this loop. This modification gained a very significant speedup due to the large ocean area over the Earth's surface. Similar strategies were applied for the rest of the code, such as the land, the present-day and ancient. Since the data structures inside those loops were complicated, and inside each loop, several subroutines were called with many different types of variables and arrays, it was critical to distinguish PRIVATE VARIABLES from SHARED VARIABLES. To use Parallel SECTIONS directives was straight forward once regions, which could be executed independently at the same time, were found. For most global summations in the code, they were modified by PARALLEL DO REDUCTION directives.

To some extent, it is easy to make mistakes and hard to debug them when OpenMP directives are used if you do not fully understand how the data moves in a shared-memory system. Therefore, code verification and validation are necessary. Each module in the code was fully tested, integration testing and comprehensive performance evaluation were performed along with correlation with theoretical performance estimates. Critical code verification activities such as regression testing, code review, and algorithmic convergence experiments were conducted. Our final model was benchmarked by comparing with the results of the sequential code for a small scale problem. The following examples illustrate how to use OpenMP directives to parallelize the original sequential code:

```
Example 3: parallel do

! The original code
        .
        do 1008 w=mpara+1,ndata
        TMP=1.d0/sqrt(Cdd(w))
        data(w)=data(w)*TMP
        do 1008 v=1,mpara
1008    obs(w,v)=obs(w,v)*TMP
        .
! The new code
        .
!$OMP PARALLEL DO DEFAULT(SHARED)
PRIVATE(w,TMP)
        do 1008 w=mpara+1,ndata
        TMP=1.d0/sqrt(Cdd(w))
        data(w)=data(w)*TMP
        do 1008 v=1,mpara
1008    obs(w,v)=obs(w,v)*TMP
!$OMP END PARALLEL DO
        .
-----------------------------------
Example 4: parallel sections
! The original code
       .
       call smgrld(geoidt,......)
       do 216 u=1,ll-l1
       obs(u,m+n)=geoidt(u)*cons5
 216   continue
       call smdisc(upe, ......)
       do 218 u=1,nu
 218   obs(ll-l1+u,m+n)=upe(u)*rate
```

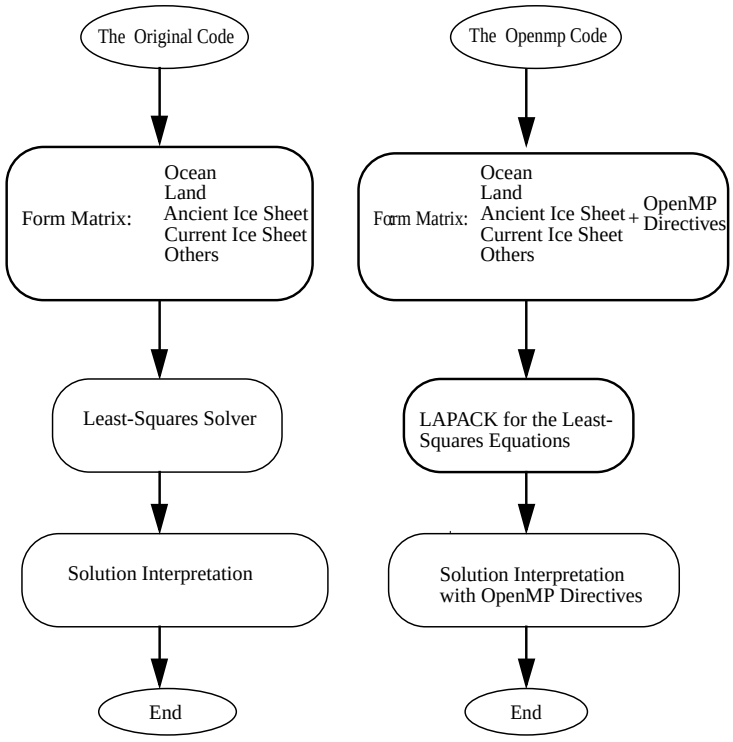

Fig. 1. High-level framework of the global inverse model illustrating major software components for both (a) the original model, and (b) the new OpenMP model.

```
       call smdisc(vhe,.....)
       do 220 u=1,nalt
 220   obs(ll-l1+nu+u,m+n)=vhe(u)*rate
       .
! The new code
       .
!$OMP PARALLEL SECTIONS
!$OMP SECTION
       call smgrld(geoidt,......)
       do 216 u=1,ll-l1
       obs(u,m+n)=geoidt(u)*cons5
 216   continue
!$OMP SECTION
       call smdisc(upe, ......)
       do 218 u=1,nu
 218   obs(ll-l1+u,m+n)=upe(u)*rate
!$OMP SECTION
       call smdisc(vhe,......)
       do 220 u=1,nalt
 220   obs(ll-l1+nu+u,m+n)=vhe(u)*rate
!$OMP END PARALLEL SECTIONS
       .
-----------------------------------
```

```
Example 5: parallel do reduction
! The orignal code
       .
       do 830 n=mp1,mp2
       tdata(k)=tdata(k)+obs0(k,n)*
       tpara(n)
830    datam(k)=datam(k)+obs0(k,n)*
para(n)
       .
! The new code
       .
!$OMP PARALLEL DO REDUCTION(+:
tdata(k), datam(k))
      do 830 n=mp1,mp2
      tdata(k)=tdata(k)+obs0(k,n)*
      tpara(n)
 830  datam(k)=datam(k)+obs0(k,n)*
 para(n)
!$OMP END PARALLEL DO
       .
```

After all this optimization, parallelization, verification, and validation, extensive performance tests were

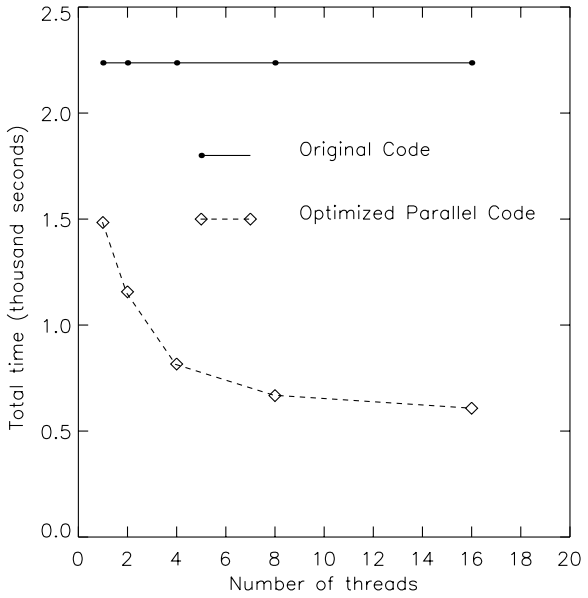

Fig. 2. The real time comparison between the optimized parallel code and the original code.

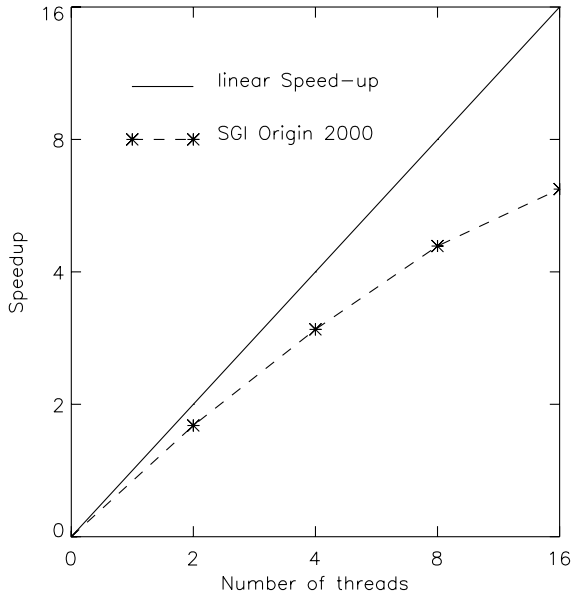

Fig. 3. Speed up of the parallel global inverse model on the SGI Origin 2000 which do not include calls to LAPACK.

carried out on the Origin 2000. This allows us to fully understand the scalability of the code and to run large-scale applications with a large number of threads to attack scientific problems. Detailed code performance data are given in the following section.

6. Performance and results

For the parallel version of the global inverse model, timing tests were performed on the SGI Origin 2000. Table 1 shows the real time comparison between the optimized code and the original code for a test case with a three degrees global model on a single processor. A global grid with a horizontal resolution of 3 degrees was used, and the global surface was divided into 4682 grid cells with spatial scales of about 300-km. The total wallclock time is significantly reduced by using optimization techniques and a well-optimized scientific library. Figure 2 shows the real time comparison between the optimized parallel code and the original code for a smaller test case with a horizontal resolution of 8 degrees using different numbers of threads. The total wallclock time is significantly reduced by using optimization techniques and multiple threads. The parallel code gives good speedup vs. the number of threads, but the efficiency decreases when more threads are used.

In order to find out what kinds of factors affect the code efficiency, several timers inside the entire code were set up and the performance of each module was measured. It turned out that when the code called LAPACK subroutines, there was no speedup gained inside LAPACK when the number of threads increased. This library can be called by multi-threads at the same time but for each subroutine of LAPACK the use of multiple threads does not improve its performance. This is because that currently, there is no directive-implemented LAPACK library available on the SGI Origin 2000. Our code did not gain any speedup with multi-threads when LAPACK was called, but as we discussed earlier, the LAPACK subroutines did give an excellent performance comparing with user's original least-squares solver on a single thread. Figure 3 shows the timing of the global OpenMP code without using the LAPACK subroutines. It gives good speedup vs the number of threads. These data indicate that users can achieve good parallel performance with a relatively small amount of work (a couple of lines need to be added for each subroutine) through the effective use of OpenMP directives. Since the code did not gain significant improvement of the performance by calling LAPACK using multi-threads and the least-squares solver is one of the major parts in the code, to use an alternative paradigm MPI should improve its overall performance for the entire code by benefiting from the version of the LAPACK on distributed-memory systems. In the future, if LAPACK allows users to use any number of threads inside LAPACK subroutines, our code performance will be

Table 1
Single PE performance data of the optimized code and the original code in seconds

Module	The optimized code	The original code	Speed-up
Matrix formation and Interpretation	9913	13656	35%
Least square equation solver	4926	22368	450%
Entire code	14839	36024	240%

significantly improved by calling multi-threads inside the least-squares solver of LAPACK.

One more issue we would like to discuss about our OpenMP code is that the code is limited to using a moderate number of threads for high resolution applications due to the memory limitation on the SGI Origin 2000 at JPL. We could not run a large number of threads with high resolution grids. If the global surface of the Earth is divided into 10502 grid cells with spatial scales of about 200-km, the code will exhaust all system memory when 32 threads are applied as the total amount of memory for all local variables with each thread will increase when more threads are used. In other words, for those applications which require a large amount of memory, the OpenMP model might have limitations because of the memory consumption problem. Again, the MPI model might be served as an alternative paradigm which can use domain decomposition techniques and overcome the memory consumption problem. If the MPI programming model is applied, the global surface will be divided into many sub domains, and each computing node will work on one or several sub domains. In this case, each grid point is only stored in one node and there is no memory copying problem. Another alternative solution for the memory problem is to run the code on shared-memory systems with a large memory size. We did successfully run the code with a high resolution application using 200-km scale grids on a cluster of Sun workstation with 16 threads. As we discussed earlier, a MPI model does need some additional programming, and an OpenMP model will result in a fast parallel version of a sequential code. Both programming models can be chosen according to different applications.

Once our parallel OpenMP code was ready for production runs, many numerical experiments were carried out to achieve our scientific objectives. We investigated the expected contributions of secular gravity and topography change measurements to the determination of present-day and historical ice mass variations. In the simulation, anticipated measurement accuracies and resolution for the GRACE gravity and ICESat altimetry missions are used. Running our parallel global simultaneous grid inverse algorithm code on parallel systems, we were able to assess separation of global sources of mass variation and to statistically evaluate spatio-temporal resolution and accuracy from full posterior covariance matrices. Linear solutions for a 200-km scale global grid were achieved. Although, these results are still preliminary, it clearly demonstrates the great potential for applying this approach to solving problems in realistic global geometries using parallel systems with multi-threads.

7. Discussion and conclusions

In the present study, we have successfully ported a sequential global inverse code to a shared-memory system. After the entire code is optimized by using several efficient strategies, a parallel global inverse code has been designed using the OpenMP programming model. This is the first optimized parallel version of a global inverse code. It gives good speedup, which will reduce the total processing time for the GRACE and the ICESat secular data and some other applications as well. The code scales fairly well, and achieves reasonable speedup in performance as the number of threads increases. It can be easily ported to any shared-memory parallel system which supports the OpenMP programming model. With a relatively small amount of effort, our performance data on the SGI Origin system gave a good example of shared-memory computing applications with OpenMP directives. For the memory consumption problem, which is associated with the usage of a large number of threads for high resolution applications, to use the SGI Origin 2000 or other shared-memory systems with a relatively larger memory size will improve the situation. In the future, if LAPACK allows users to control the number of threads which can be used inside of LAPACK subroutines, the performance of our current code will be improved by applying multiple threads into the least-squares solver. The parallel code has been tested with various input data sets, and the present results illustrated here clearly demonstrate the potential for applying this approach to large-scale scientific applications. Numerical solutions for more detailed non-linear inverse problems are also under investigation.

Acknowledgments

The research described in this paper was performed at the Jet Propulsion Laboratory (JPL), California Institute of Technology, under contract to the National Aeronautics and Space Administration. The SGI Origin 2000 Supercomputers used to produce the results in this paper were provided with funding from the NASA offices of the Earth Science, Aeronautics, and Space Science.

References

[1] E. Anderson et al., LAPACK Users' Guide, www.netlib.org, 1999.
[2] OpenMP Architecture Review Board. Openmp fortran application program interface, version 2.0, 2000.
[3] J.O. Dickey et al., *Satellite Gravity and the Geosphere: Contributions to the study of the solid Earth and its uid envelope,* National Academy Press, Washington, DC, 1997.
[4] W. Gropp, E. Lusk and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface,* MIT Press, 1999.
[5] D. Han and J. Wahr, The viscoelastic relaxation of a realistically stratified earth, and a further analysis of postglacial rebound, *Geophys. J. Int.* **120** (1995), 287–311.
[6] S.S. Jacobs, H.H. Hellmer, C.S.M. Doake, A. Jenkins and R.M. Frolich, Melting of ice shelves and the mass balance of antarctica, *J. Glaciol.* **38** (1992), 375–387.
[7] J.X. Mitrovica, A.M. Forte and M. Simons, A reappraisal of postglacial decay times from richmond gulf and james bay, *Geophys. J. Int.* **142** (2000), 783–800.
[8] P. Wang, D.S. Katz and Y. Chao, Optimization of a parallel ocean general circulation codel, in the proceedings of the Super Computing 97 (Awarded the Best Paper prize), 1997.
[9] X. Wu, M.M. Watkins, E.R. Ivins, R. Kwok and P. Wang, Toward global inverse solutions for the determination of current and past ice mass variations: Contribution of secular satellite gravity and topography change measurements, submitted to *J. Geophysical Research,* 2001.