

Development and Performance Analysis of a Simulation-Optimization Framework on TeraGrid Linux Clusters

Baha Y. Mirghani¹, Michael E. Tryby¹, Derek A. Baessler², Nicholas Karonis²,
Ranji S. Ranjithan¹ and Kumar G. Mahinthakumar¹

¹Department of Civil & Environmental Engineering
North Carolina State University
Raleigh, NC 27695, USA

²Department of Computer Science
Northern Illinois University
DeKalb, IL 60115, USA

Abstract. A Large Scale Simulation Optimization (LASSO) framework is being developed by the authors. Linux clusters are the target platform for the framework, specifically cluster resources on the NSF TeraGrid. The framework is designed in a modular fashion that simplifies coupling with simulation model executables, allowing application of simulation optimization approaches across problem domains. In this paper the LASSO framework is coupled with a parallel groundwater transport simulation model. The performance of the LASSO framework is measured using a source history reconstruction problem and benchmarked against an existing MPI based implementation developed previously. Performance results indicate that communication overhead in the LASSO framework is contributing significantly to wall times. The authors purpose and will conduct several performance optimizations designed to ameliorate the problem.

1 Introduction

“Simulation optimization” is a general term used to describe a family of optimization techniques which utilizes simulation models for the evaluation of objective and constraint functions. A wide variety of applications lend themselves to simulation optimization techniques, including engineering design optimization, optimization of stochastic systems, model calibration, and solution of inverse problems. The LASSO framework is currently under development by the authors [9], and has been implemented using object oriented programming (OOP) principles in the Java programming language.

Many simulation optimization techniques exhibit a coarse grained parallel structure that requires many uncoupled simulation model evaluations. When the simulation model is itself written for parallel execution two levels of parallelism, coarse grained with in the optimization algorithm and fine grained within the simulation model, are present. LASSO is designed to exploit both coarse and fine grained parallelism for efficient parallel speed up.

The LASSO framework is portable and tested on several Linux Cluster configurations; however, the target platform for LASSO is the NSF TeraGrid. The TeraGrid is

a distributed computing infrastructure providing peta scale data collection, analysis, and storage; and tera scale computational capabilities. The nine TeraGrid sites themselves are a heterogeneous mixture of computational resources, consisting of cluster, SMP and SP nodes. The core Linux cluster resources available on the TeraGrid originate from the NSF Distributed Terascale Facility (DTF) and are located at NCSA, SDSC, CalTech and UC/ANL. LASSO accesses these TeraGrid cluster resources via MPICH-G2, the Globus Toolkit (TM) and the Java Cog Kit.

The LASSO framework has been developed as a generic search and optimization tool useful across many application areas. The framework currently supports evolutionary search heuristics (genetic algorithms and evolutionary strategies), direct search techniques (Nelder-Mead Simplex, Hooke and Jeeves, and Powell's conjugate directions). Development of gradient based procedures, hybrid search techniques, and advanced distributed genetic algorithms are planned.

An important requirement guiding the development of LASSO is scaling efficiency. This paper documents a battery of performance tests used to study the limitations of the current framework architecture, and guide our continuing development efforts.

2. Application

The application used in this study was the source release history reconstruction problem [8]. Reconstructing the groundwater releases history from available concentration measurements is an inverse problem. Solving this problem is important in environmental forensics, where liabilities and responsibilities are forced to potential polluters. Generally, in "real world," problems the containment locations are known, but the contaminants release time histories are unknown.

In this application, a parallel ground water transport model was coupled with LASSO to simulate ground water transport phenomena. The source history reconstruction problem is formulated as a simulation optimization problem solved using a genetic algorithm. The sections that follow are a brief technical description of the simulation model and optimization problem used in the performance analysis.

2.1 Simulation Model -- PGREM3D

PGREM3D stand for Parallel Groundwater transport and REMediation codes, a suite of massively parallel codes used for numerical simulation of three-dimensional groundwater transport and remediation problems [5]. The codes based on the finite element methods. PGREM3D contents of two modules, flow and transport code. The flow module solves the steady state groundwater flow equation that described by [1].

$$\nabla(K\nabla h) = q \quad (1)$$

where h is the computed head field from the groundwater flow equation, K is the 3x3 hydraulic conductivity tensor (usually diagonal), θ is the porosity, and q represents the source/sink terms coming from injection/pumping wells. The hydraulic head

field h is computed from equation (1), then velocity field v will be computed the from Darcy's law

$$\theta v = -K \nabla h \quad (2)$$

2.1.1 The Transport Governing Equations The system of equations describing the transport of nc dissolved components undergoing reactions in saturated porous media is defined by

$$\frac{\partial C_i}{\partial t} = \nabla \cdot (D \cdot \nabla C_i) - \nabla \cdot (C_i v) + \frac{q}{\theta} (C_i - C_{0i}) - R_i \quad i = 1, 2, 3 \dots nc \quad (3)$$

where v is the 3x1 velocity field vector, D is the 3x3 dispersion tensor dependent on v , and C_i is the dissolved concentration of component i . The term $q(C_i - C_{0i})/\theta$ represents the source term with volumetric flux q , medium porosity θ , and injected concentration C_{0i} . R_i is the rate of mass loss of component i due to reactions. The elements of the 3x3 dispersion tensor D are defined by

$$D_{ij} = a_T |v| \delta_{ij} + (a_L - a_T) \frac{v_i v_j}{|v|} + D_m \quad (4)$$

where a_L and a_T are longitudinal and transverse dispersivities assumed to be constant, D_m is the coefficient of molecular diffusion assumed to be constant (usually very small), and d_{ij} is the Kronecker delta (if $i=j$, $d_{ij}=1$ else $d_{ij}=0$).

The reaction term R_i for components undergoing linear kinetic sorption, first order decay and biodegradation reactions is given by

$$R_i = B_i + (\lambda_i + \lambda_{w_i}) C_i + \frac{\rho}{\theta} \frac{\partial S_i}{\partial t} \quad i = 1, 2, 3 \dots nc \quad (5)$$

where B_i is the rate of mass loss due to biodegradation reactions, λ_i is the radioactive decay coefficient, λ_{w_i} is the aqueous phase first order decay coefficient and the term $\partial S_i / \partial t$ is the rate of change of component i in the sorbed phase. More detailed description is available at [5].

2.1.2 Parallel Implementation The numerical methThe numerical methodology used in modules is the Galerkin finite element method with eight-node linear hexahedral elements. A logically rectangular grid structure is assumed but irregular geometries are supported using distorted elements. A Crank-Nicolson approximation is employed for the time derivative terms. A lumped mass formulation [2] is used for all time-derivative and non-derivative (zeroth spatial derivative) terms. The coupled non-linear system is solved using a modified form of the Sequential Iterative Algorithm (SIA). Several Krylov subspace iterative solvers are implemented in the code for the matrix solution [3].

This transport module simulator is parallelized using a two-dimensional domain decomposition (in the x and y directions). Explicit message passing interface library

(MPI) was utilized to exchange information between these domains. The codes are written in Fortran using double-precision arithmetic. The simulator has been tested extensively for scalability and performance on different parallel architectures [4] and [6] However in this study the TeraGrid supercomputers at NCSA and SDSC were used widely.

2.2 Source History Reconstruction Problem

The hypothetical domain with a single source utilized in this study is illustrated in Figure 1. The problem assumes that the contaminant source locations are known, but that the contaminant release histories at the sources are unknown. Concentration observations (at the 18 monitoring wells) are collected periodically at the vertical cross-sections of the domain (see Figure 1), and generating a concentration time series at each monitoring location. We parameterize a general concentration time series as follows;

$$\mathbf{C} = [c(0), c(\Delta t), c(2\Delta t), \dots] \quad (6)$$

where c (M/L^3) is a concentration and Δt is a periodic time interval. We wish to reconstruct the source release history over a finite time horizon tr extending from the time in the past when monitoring activities started t_{s_0} towards the present, and referred to as the release history reconstruction period, with a periodic interval Δtr . Monitoring activities are conducted at n_m monitoring wells. The number of samples taken at a monitoring well n_s over the release history reconstruction period is equal to $(tr/\Delta tm)n_m$, where Δtm is the periodic monitoring interval. In the general case, t_0 and t_{s_0} do not correspond; however, for the problem being studied here we assume that sampling activities start at t_0 .

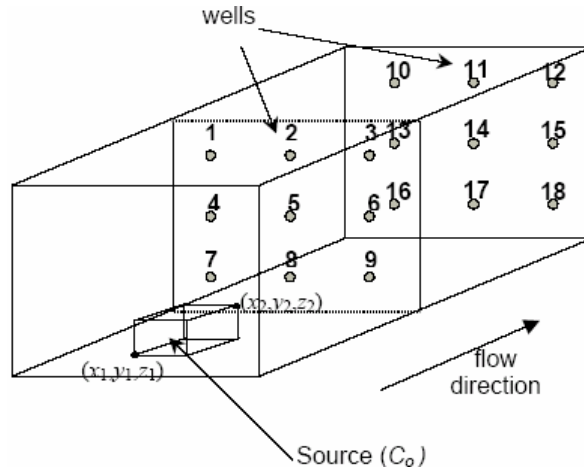


Fig. 1. 3D Hypothetical domain with a single source and observations wells [8]

We attempt to model the observed concentrations using the PGREM3D simulation, which provides the relationship $Cm = f(Cr)$, where Cr is the source release concentration time series and Cm is the time series of modeled monitoring concentrations. Thus we are faced with the solution of an inverse problem. The inverse problem is posed as an optimization model where the objective is the minimization of the root square error (RSE) between the observed and calculated concentrations;

$$\min_{Cr_j} \sqrt{\sum_{i=1}^n (Co_i - Cm_i)^2} \quad (7)$$

where the concentration time series Co_i , Cm_i are observed and modeled concentrations respectively and $i = 1 \dots n_m$ is the index of monitoring well locations. The time series of released concentrations Cr_j , where $j=1 \dots n_r$ is the number of sources, are the set of decision variables over which the problem is solved. Constraints are written to enforce decision variable positivity.

subject to:

$$Cr_j > 0, \quad \forall j = 1 \dots n_r \quad (8)$$

Depending on the number of contaminant sources, the number of decision variables is equal to the product of the number of contaminant sources times the number of time durations.

3 Timing Study Experiments

Timing studies were conducted to measure the performance of the current LASSO implementation and identify performance bottlenecks to be addressed in ongoing development. The release history reconstruction problem described above is the target application used for the performance analysis, and is solved here using a genetic algorithm (a stochastic search heuristic). Genetic algorithms are inherently parallel by their nature, that is to say many independent simulation model evaluations are required to find the best solution. The PGREM3D model is itself written for parallel evaluation. Thus, for maximum efficiency, the solution procedures implemented in LASSO must effectively manage both the coarse grained task parallelism of the genetic algorithm and the fine grained data parallelism of the simulation model.

Sayeed et al. 2002 developed an Efficient Parallel Optimization (EPO) framework implemented in Fortran MPI for solving problem exhibiting multilevel parallelism like that present in the problem we describe here [7]. The software design approach taken, however, resulted in tight coupling between the search algorithm and the simulation model. This resulted in a highly efficient implementation, but one that makes it difficult to couple with other simulation models. Here we strive for a generic search framework which is loosely coupled with the simulation model, yet efficient. The EPO implementation is used here as a performance reference. Our intention is not an outright performance comparison between Java and Fortran MPI, since we anticipated Java to be slower in the first place. Rather, we wish to compare the two implementa-

tions and gauge performance improvements in LASSO relative to conventional high performance benchmark.

For this application, the grid resolution for the simulation model resulted in 17,391 finite element nodes within the ground water domain, and the simulation duration was 1000 time steps. The source history reconstruction problem was formulated with 18 monitoring wells sampled at a frequency, Δtm , equal to 10 times the simulation model time step. Thus, a total of 1800 observations are used corresponding to 18 wells and 100 periodic samples. The release history reconstruction period, tr , was equal to the simulation duration, and the reconstruction period frequency, Δtr , was equal to 100 times the simulation model time step. Thus, there were 10 decision variables in the optimization problem.

The population size for the LASSO and the EPO genetic algorithms was 128 and the algorithms advanced for 10 generations before times were taken. Thus, the theoretical number of simulation model evaluations for each is 1280. The other genetic algorithm settings used in LASSO and EPO are not documented here. Every attempt was made to make the benchmark performance meaningful; this includes selection of search algorithms, settings, and decision variable encodings.

The timing studies presented here were conducted on the NCSA and SDSC IA-64 Itanium clusters on the TeraGrid. Scalability runs were performed at each site by successively doubling the number of processors available to the computation. The PGRM3D executable was compiled and run in a “uniproc” configuration (running on a single processor). This enabled us to isolate the coarse grained parallelism within the solution procedure for performance benchmarking.

4 Results

When studying our results we realized that the LASSO and EPO genetic algorithm implementations are quite different. The theoretical number of simulation model evaluations for each implementation should have been 1280. The actual number of simulation model evaluations performed, however, was 938 and 1280 for the LASSO and EPO implementations respectively. These differences can be attributed to a performance optimization made in LASSO which eliminates duplicate evaluations from occurring between generations. The results presented in Tables 1 through 4 display both the total wall time and the wall time normalized by the number of evaluations performed, while Figures 2 and 3 display data normalized by the number of evaluations.

Scalability results are shown in Figure 2. Scalability across implementations and across TeraGrid sites was comparable up to 32 processors, after which performance of the LASSO implementation faltered. At both the SDSC and NCSA sites, the EPO implementation scaled better than LASSO. In general both LASSO and EPO results at SDSC were slower than those from NCSA (see Tables 1 through 4 and Figure 2), though this effect is more pronounced for the LASSO implementation. The SDSC and NCSA clusters are built on identical microprocessor technology, however, differences in administration complicated porting between sites, and may have contributed to performance differences.

Table 1. Wall time and scalability results for LASSO on TeraGrid NCSA site

Number of Processors	Time (hh:mm:ss)	Time (sec)	Time for one Eval per Processor (sec)	Scalability
1	5:19:46	19186	20.45	1.00
2	2:43:53	9833	20.97	1.95
4	1:21:51	4911	20.94	3.91
8	0:42:17	2537	21.64	7.56
16	0:22:09	1329	22.67	14.44
32	0:11:51	711	24.26	26.98
64	0:07:54	474	32.34	40.48
128	0:05:18	318	43.39	60.33

Table 2. Wall time and scalability results for EPO framework on TeraGrid NCSA site

Number of Processors	Time (hh:mm:ss)	Time (sec)	Time for one Eval per Processor (sec)	Scalability
1	5:49:14	20954	16.37	1.00
2	2:55:59	10559	16.50	1.98
4	1:30:59	5459	17.06	3.84
8	0:46:32	2792	17.45	7.51
16	0:24:33	1473	18.41	14.23
32	0:12:12	732	18.30	28.63
64	0:07:14	434	21.70	48.28
128	0:03:18	198	19.80	105.83

Wall time normalized by the number of evaluations is shown in Figure 3. The timings for one evaluation per processor for EPO varied from 15 to 20, while for LASSO it varied from 20 to 60. LASSO is consistently slower than the EPO implementation. This effect becomes more pronounced as the number of processors increases. This leads us to conclude that communication contention increases detrimentally in the LASSO implementation. And that the rate of this increase is proportional to the number of processors. Therefore the most fruitful performance optimizations may be found in the master worker communications and in the model execution mechanisms.

Table 3. Wall Time and scalability results for LASSO on TeraGrid SDSC site

Number of Processors	Time (hh:mm:ss)	Time (sec)	Time for one Eval per Processor (sec)	Scalability
1	7:00:20	25220	26.89	1.00
2	3:40:13	13213	28.17	1.91
4	2:08:06	7686	32.78	3.28
8	1:09:15	4155	35.44	6.07
16	0:36:12	2172	37.05	11.61
32	0:20:49	1249	42.61	20.19
64	0:08:49	529	36.09	47.67
128	0:08:14	494	67.41	51.05

Table 4. Wall time and scalability results for EPO framework on TeraGrid SDSC site

Number of Processors	Time (hh:mm:ss)	Time (sec)	Time for one Eval per Processor (sec)	Scalability
1	5:25:03	19503	15.24	1.00
2	3:03:35	11015	17.21	1.77
4	1:30:07	5407	16.90	3.61
8	0:45:56	2756	17.23	7.08
16	0:23:29	1409	17.61	13.84
32	0:11:54	714	17.85	27.32
64	0:06:09	369	18.45	52.85
128	0:03:24	204	20.40	95.60

5 Conclusion

From the results we draw the following conclusions:

- Communications overhead contributes significantly to evaluation times especially when the numbers of processors was greater than or equal to 32.
- File I/O between the execution servers and their simulation executables make model evaluations more costly in the LASSO Framework.
- Though the TeraGrid sites utilized in these tests are homogeneous with respect to microprocessor technology, system configuration differences at the two sites contributed to performance differences which we document in our results.
- The ratio of Java/Fortran wall times in this study ranged from 1.2 to 3.2, we argue that this is an acceptable level of slow down given the advantages of the LASSO framework design.

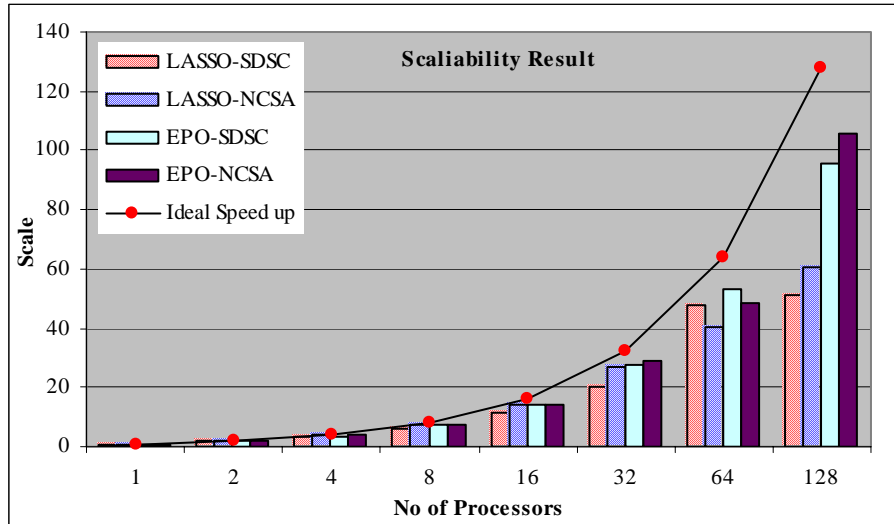


Fig. 2. Normalized scalability results for LASSO and EPO frameworks on NCSA and SDSC TeraGrid sites

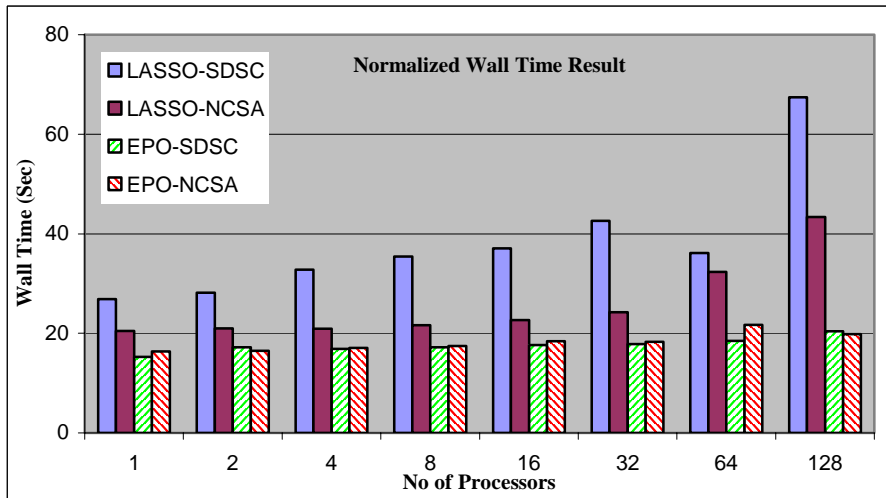


Fig. 3. Normalized wall time results for LASSO and EPO frameworks on NCSA and SDSC TeraGrid sites

6 Improvements to the Current Implementation

Many factors could be contributing negatively to LASSO performance. The performance benchmark results analyzed here suggest that communication overhead may be

contributing significantly to runtimes. Here we propose performance optimizations targeting communications to and from LASSO evaluation servers and between the evaluation server and the PGREM3D transport executable.

LASSO maintains a task pool of evaluations and distributes them using a simple master worker strategy. The operation of the task pool is currently handled on a first in first out basis by a single thread of execution. Possible performance optimizations include maintaining multiple task pools, multi-threading task pool operations, or handling tasks as they are completed. Java TCP-IP socket communications are currently used between the LASSO and its evaluation servers, and furthermore, these communications are occurring over the clusters administrative Ethernet network. Further performance gains could be realized by directing these communications over the Myrinet high speed network.

LASSO sends decision variables and receives objective and constraint results via a file exchange between the execution server and the PGREM3D executable. We propose eliminating file I/O and creating a generic interface for decision variable and evaluation result exchange using MPICHG2 (see Figure 4). The approach we propose involves the construction of an adaptor process written in the C programming language that translates Java objects into native variables compatible for MPICHG2, and sends them via a socket connection with the executable established via an MPICHG2 call.

We plan to include the results of these performance optimizations at the LCI Conference presentation.

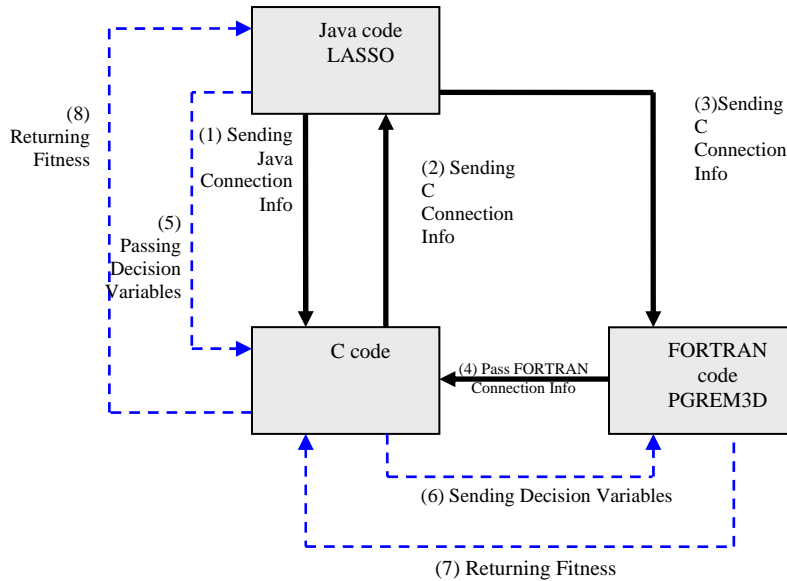


Fig. 4. LASSO framework using MPICHG2

References

1. Bear (1979), *Hydraulics of Groundwater*, McGraw-Hill, 1979.
2. Huyakorn, P.S., and G.F. Pinder (1983). *Computational Methods in Subsurface Flow*, Academic Press, New York, N.Y.
3. Mahinthakumar, G., F. Saied, and A. J. Valocchi, (1997). Comparison of some parallel krylov solvers for large scale contaminant transport simulations, High Performance Computing 1997 (Editor: A. M. Tentner), p. 134-139, 1997.
4. Mahinthakumar G., and Saied F. (1999). *Implementation and Performance Analysis of a Parallel Multicomponent Groundwater Transport Code*, CD-ROM Proceedings of the 1999 SIAM Parallel Processing Meeting, San Antonio, TX, March 1999.
5. Mahinthakumar, G., (1999). *PGREM3D: Massively Parallel Codes for Groundwater Flow and Transport*, Unpublished Report <http://www4.ncsu.edu/~gmkumar/pgrem3d.pdf> .
6. Mahinthakumar, G., and Saied, F., (2000). *Hybrid MPI-OpenMP implementation of an implicit finite element code*, High performance computing 2000, Washington, DC, April 10-13, 2000.
7. Sayeed, M. and G. Mahinthakumar, (2002). *A multilevel parallelization scheme based on MPI communicators for solving groundwater inverse problems*, High performance computing 2002, Society for Computer Simulation International, p. 137-144.
8. Sayeed, M. (2003). *An efficient parallel optimization framework for inverse problems*. PhD dissertation submitted to Department of Civil. Construction and Environmental Engineering, North Carolina State University, Raleigh, USA.
9. Tryby M. (2004) Large Scale Simulation Optimization (LASSO) Framework User's Manual, unpublished.