

A software system for scalable parameter estimation on clusters

Tom Bulatewicz, Daniel Andresen, Stephen Welch^a, Wei Jin^a, Sanjoy Das^b, Matthew Miller

Department of Computing and Information Sciences

Kansas State University

234 Nichols Hall, Manhattan KS, 66506, USA

{tombz, dan, welchsm, jacking, sdas, millermj}@ksu.edu

^aDepartment of Agronomy

^bDepartment of Electrical and Computer Engineering

Abstract

Advancements in data collection and high performance computing are making sophisticated model calibration possible throughout the modeling and simulation community. The model calibration process, in which the appropriate input values are estimated for unknown parameters, is typically a computationally intensive task and necessitates the use of distributed software components. These components are often heterogeneous due to the combination of the model and the optimization software, making scalability difficult to achieve. We have developed a hybrid software system for parameter estimation consisting of an optimization algorithm implemented in a mathematical scripting language, a legacy Fortran model, and an MPI client program. Through a series of optimizations, we achieved near-linear speedup when the model is executed as a standalone process, and achieved super-linear speedup when the model is executed as a subroutine. We report on our optimization techniques and performance results of an estimation problem within the context of an ongoing modeling study.

1 Introduction

Parameter estimation seeks to identify the unknown values of model parameters that provide the best fits, most commonly in terms of minimum errors or maximum likelihood, between predictions and observations [10]. In general, this requires the solution of a nonlinear optimization problem which is often a computationally intensive process. Nonetheless, parameter estimation is becoming increasingly practical. Advances in data collection methods have resulted in a wealth of data making model calibration possible for

models or study sites that previously lacked sufficient data. The increasing power and availability of clusters is making more computationally intensive and sophisticated optimization techniques possible. As a result, there is an increasing need for software to support flexible and efficient parameter estimation.

This is the case in the consortium for Global Research on Water-based Economies (GRoWE) at Kansas State University where we are employing high performance computing (HPC) and a series of multidisciplinary models to study the agricultural, hydrological, economic, and sociological impacts of the decline of the Great Plains Aquifer in western Kansas. We are using the Environmental Policy Integrated Climate (EPIC) model [17] to simulate the agricultural processes that draw upon the aquifer. Before we can use the model for predictive simulation, we must calibrate it to our initial study site of Sheridan County, Kansas. The strong interdisciplinary nature of the consortium and the practical need for software reuse makes the development of a homogeneous software system for parameter estimation impractical. Researchers in different disciplines must be free to use the methodologies and tools (both new and existing) that are customary in their particular field. As a result, the system is composed of heterogeneous and distributed components making scalability difficult to attain. In this work we report on our efforts to meet the challenge of achieving scalability within a heterogeneous software environment on a Beowulf cluster.

We review related work in the following section and describe the parameter estimation approach and implementation in Section 3. In Section 4 we present the results of our efforts, and give our conclusions and current work in Section 5.

2 Related Work

In recent decades, a number of gradient-based local optimization software tools have been successfully applied to parameter estimation problems. Two popular tools are PEST [6] and UCODE [14]. In the case of problems that require global optimization, the successful application of a variety of methods has spurred the development of parallel parameter estimation frameworks such as PGO [11] and Nimrod/O [1]. Such frameworks simplify the task of performing a parameter estimation experiment at the cost of reduced flexibility. This restricted flexibility, such as the lack of support for high-speed interconnects, can limit the efficiency of the software in an HPC environment. Achieving maximum performance requires the software to be tuned to the specific hardware environment within which it will be used. There has been a variety of HPC software created to support various hardware environments, such as High-Performance Fortran [15], MPI [9], OpenMP [5], Globus [7], Condor [16], and the Berkeley Open Infrastructure for Network Computing (BOINC) [2]. These are designed to support scalability only within the bounds of their own software environment. It is the responsibility of the user to ensure efficiency when heterogeneous software components are used in conjunction with these systems, such as components written in different languages, or for different platforms. There has been considerable work in the area of component compatibility and interface design, such as the Common Component Architecture (CCA) [3], that resolves differences between component languages. In the case of highly heterogeneous components, though, some of which may be standalone third-party applications, adherence to a standard interface introduces additional overhead both in terms of development effort and performance. We have developed a software system to serve as an experimental optimization workbench for parameter estimation that offers flexibility through a heterogeneous combination of components with a scalable implementation on clusters.

3 Method

3.1 Parameter Estimation Approach

We are calibrating the Environment Policy Integrated Climate (EPIC) model for our initial study site of Sheridan County, Kansas. The EPIC model simulates a variety of biophysical processes including crop growth, hydrology, erosion, and fertilization, as well as management practices, such as planting/harvest dates and

irrigation schedules. As a result, the model relies upon a large number of input parameters. Commonly accepted values can be reliably used for many of these, but under the particular conditions of this study we felt it important that some of them be estimated. The 10 estimated parameters fall into two groups: (1) plant growth and development: optimum and minimum temperature for plant growth, energy-to-biomass conversion ratio, fraction of growing season complete when leaf-area index starts declining, leaf-area decline rate, energy-to-biomass conversion ratio decline rate, and (2) crop management: water stress level at which to irrigate, least and most water that can be applied in one irrigation, and minimum interval between successive irrigations.

The unit of computation for crop models is the site-year, which is the simulation of the production of a crop at one location in one growing season. We parameterized EPIC such that a single execution of the model simulates a single site-year. The observed data against which the simulated results are evaluated consists of two quantities: the total annual county harvest from 1990-2000, and the amount of water pumped from each irrigation well on each year. Figure 1 illustrates the distribution of wells throughout Sheridan County. Each well represents a different site.

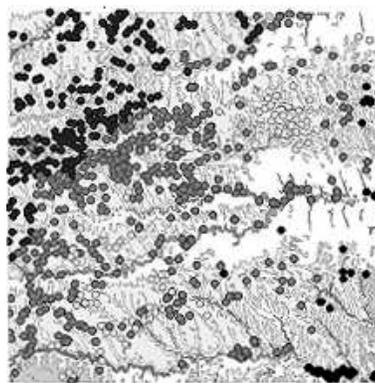


Figure 1. Clustering of wells in Sheridan County, Kansas. We selected a stratified sample from these clusters to calibrate EPIC using 1990-2000 data. [12]

In year 2000, there were 783 wells used for corn production in Sheridan County. We developed a parallel genetic algorithm that uses a population of 100 trial solutions that are allowed to evolve for 200 generations. On each of the 200 generations of the genetic algorithm, each of the 100 trial solutions is evaluated. The evaluation of a trial solution requires the simulation of each of the 11 years for each of the 783 wells. Since there are no dependencies between the evalua-

tions of the trial solutions, the entire population can be evaluated in parallel in a master-slave style [4], which requires the simulation of 8,613 site-years on each generation. The total number of site-years in a complete estimation is 17.2×10^7 . The error analyses we wish to use require up to 250 complete estimations, for a total of 43.1×10^9 site-year simulations. We report the performance of a single generation in this work, which can be used to compute runtime estimates for problems of any size. We chose a representative problem size of 577 site-years in a single generation (derived from a 10% sample of the wells chosen for our initial studies).

The objective function uses the Maximum Entropy [8] method and evaluates the fitness of a set of a trial solutions based on the predicted and observed harvest and water usage. The quality of the solution given by any nonlinear optimization process is dependent upon the specific characteristics of both the algorithm and the data. For this reason, optimization must always be considered an exploratory process utilizing multiple algorithms and parameterizations of those algorithms. Such a task demands a flexible software environment in which the users are free to experiment. We plan to implement alternative optimization algorithms in our system, including a local search optimization.

3.2 Implementation

The system consists of four software components as illustrated in Figure 2: (1) the starter program that initiates the system, (2) the genetic algorithm that generates the trial solutions, (3) the client program that executes the model in parallel for the trial solutions across a cluster, and (4) the model itself.

The process begins with the execution of the starter program which distributes all the necessary input files (1 MB) and executable files to the RAM disks of the cluster nodes. It then invokes the genetic algorithm on the RAM disk of one of the cluster nodes.

The genetic algorithm is written in SciLab, a popular mathematical scripting language. The algorithm generates the initial population of trial solutions and then begins a loop in which the population is evaluated and then updated on each iteration. The possible values of each parameter within a trial solution are restricted to a parameter-specific range that we approximated based on the model documentation and expert opinion. The algorithm evaluates a population of solutions by invoking a client program that performs the necessary simulations on the compute nodes of the cluster. The trial solutions are communicated from the genetic algorithm to the client via a file. The file specifies what the 10 estimated values are and lists each site-year that needs

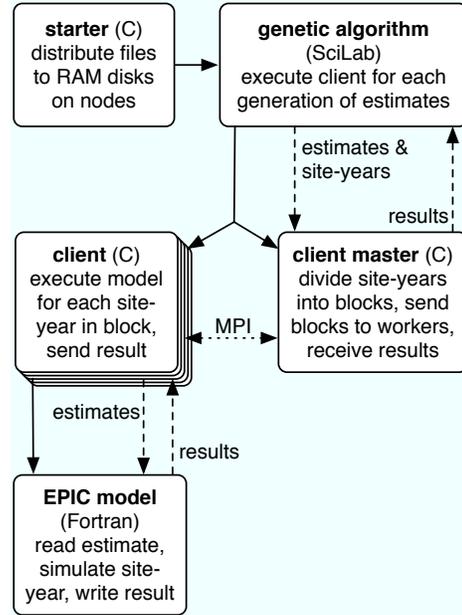


Figure 2. Software architecture. Solid lines indicate process invocations, and dashed lines indicate local communication via reading and writing files.

to be simulated. After the client completes the simulation of all the site-years (in parallel) with the given estimated values, it writes a file that lists the harvest and water use for each site-year. The file is read by the genetic algorithm and is used by the objective function to compute the error. The genetic algorithm communicates with the client via file i/o because it provides a clean and generic interface. This allows us to easily substitute an alternative client, as described at the end of Section 4.

Each time the client is invoked by the genetic algorithm, the master process reads the list of site-years and estimated parameters to be simulated and divides them into blocks of simulations that are distributed to the compute nodes via MPI. In the case of 57,700 site-years, we used a block size of $\text{ceiling}((57,700 / (\text{number of processors} - 1)) / 32)$ which ensures an even distribution of 32 blocks to each processor and avoids the case where only a few processors are completing the final blocks while the remaining processors are idle. We expect at most a single processor to be unused as the final blocks are executed. We selected 32 blocks per processor based on a series of trials we conducted with varying block sizes. We found that dividing the simulations into a single block for each processor resulted in poorer performance compared to multiple blocks per processor.

The time required by 32 blocks per processor was half of that required by a single block per processor, yet further increases in the number of blocks gave little improvement. The data communicated between the clients via MPI is $12 + 6$ (8-byte double) values per site-year, for a total of 7.9 MB per generation.

When a client process on a compute node receives a block of site-years to simulate, it steps through the list, executing the model for each site-year sequentially. Before executing the model, the client writes the 10 estimated parameter values to a file, along with the name of the input file that the model should use for the particular site-year simulation. There is a different input file for each site-year, and these are distributed to the compute nodes by the starter program. The client uses the `system` system call to invoke the model.

We modified the model source code such that it reads the 10 estimated parameter values and uses them in place of the values read from the input file. These modifications were simple and consisted of 38 lines of code. We also modified the model source code such that it writes the harvest and water use output values to a file after the simulation completes. The client reads this file and stores the outputs until all the site-years in its current block have been simulated and then sends all of the outputs to the master client process. After the master client receives all the outputs for all the blocks, they are written to a file and the client waits to receive another block.

Rather than modify the model source code, we could have enabled the client to generate a custom input file before each execution and parse the output file after each execution to obtain the harvest and water use. In this way, the model could be used within the system with no changes to its source code. Such a technique not only achieves greater model independence, but is necessary in cases where the model code cannot be modified, either due to source code restrictions or validation concerns. In this work, we chose to modify the model to avoid the overhead of reading a template input file and generating a custom input file for each simulation.

4 Experimental Results

We considered three metrics in the evaluation of the performance of the client: speedup, network bandwidth, and CPU utilization. We calculated speedup as the total clock time required by a single processor (as described above) divided by the total clock time required by 16, 32, 48, 64, 80, and 96 processors. The total clock time was measured using start/stop timestamps within the client program. The bandwidth and

CPU utilization were measured via visual inspection of cluster status graphs produced by Ganglia [13]. The CPU utilization is the average of the utilization of each participating node as reported by Ganglia. Both the total and NFS bandwidth were measured. The NFS bandwidth was assumed to be the bandwidth used by the NFS server, and total bandwidth to be the cluster bandwidth consumed as reported by Ganglia (on an otherwise idle cluster). In both cases, the average of the bytes in and bytes out were measured, which were nearly identical in all cases. Our results are summarized in Figure 3.

The onsite Beowulf cluster used in our study consists of 6 nodes, each of which has 8 dual-core AMD Opteron 2.2 GHz CPUs, for a total of 96 cores (which we will refer to as processors). Each node has 64 GB of memory and a 512 MB RAM disk. The nodes are connected to each other and to an NFS server via gigabit Ethernet. The nodes use Gentoo Linux 2.6. The EPIC model is a Fortran program compiled with `g77`, and the client is an Open MPI C program compiled with `gcc`, both of which produce no console output.

A single execution of the model requires 20.29 ms when executed on a RAM disk (real time, static link, `g77`, 100 run average, 23.64 ms dynamic link). On a physical disk, a single execution requires 214.64 ms (56.8 ms with caching effects). For this reason, the model is always executed from a RAM disk.

On one processor, the runtime would be 19.5 minutes to evaluate a single generation of 57,700 parameter estimates, and would thus require 65.0 hours for a complete parameter estimation of 200 generations. We therefore consider ideal speedup to be $0.02029 * 57,700 / \text{number of processors}$. With an ideal speedup on a cluster of 96 processors, the single generation runtime would be reduced to 12.2 seconds and the complete estimation reduced to 40.7 minutes.

Each generation of the genetic algorithm is evaluated in order, so the SciLab script and client MPI program alternate execution. The time required by the SciLab script between client executions was measured to be 2.4 seconds and increases super-linearly with the number of site-years. Improvement in the performance of the script is an area for future work. Since the genetic algorithm necessarily executes on a single node and has minimal impact on the runtime of the system, we measure scalability and performance with respect to the client.

The implementation of this system reflects a progression of techniques. Our early work focused on enabling the distributed execution of the model on a limited number of nodes with a limited amount of parallelization. We then modified the genetic algorithm to

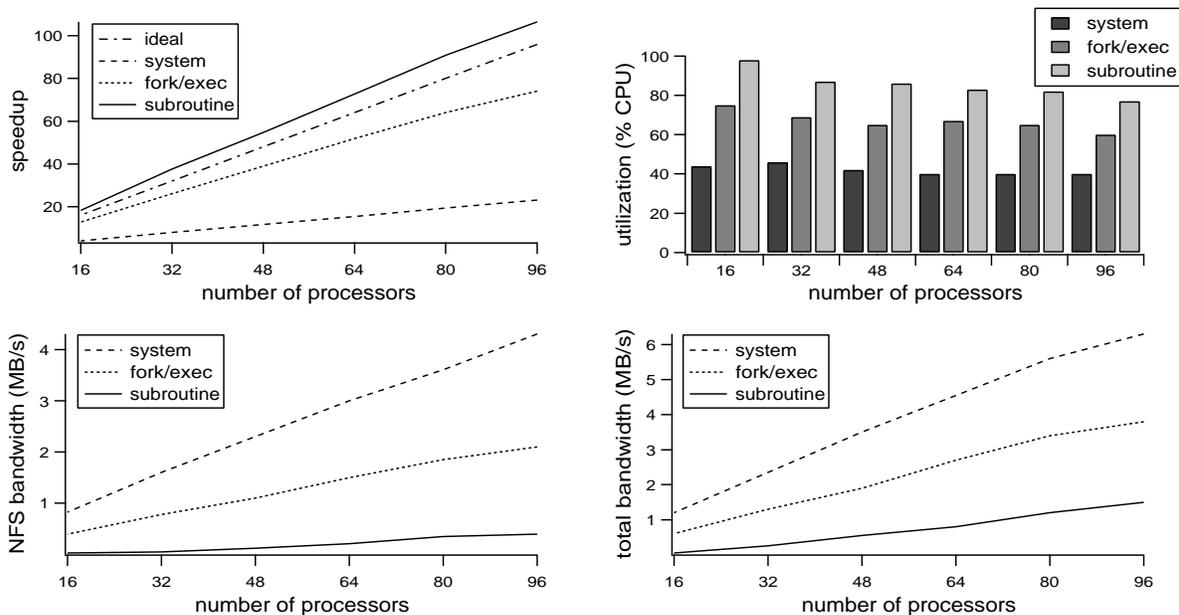


Figure 3. Performance results: speedup (top-left), CPU utilization (top-right), NFS bandwidth (bottom-left) and total bandwidth (bottom-right).

improve the potential for parallelization (by evaluating the entire population of trial solutions simultaneously, rather than individually) which allowed for the utilization of the entire cluster. We expected to achieve near-linear speedup with low bandwidth and high processor utilization due to the high level of parallelism inherent in the estimation problem. In our initial tests though, we observed a speedup of only 11.15 with 96 processors. There was twice as much system CPU (20%, averaged over the 6 nodes) than there was user CPU (10%), and 6 MB/s of bandwidth was consumed, 4 MB/s of which was NFS usage. Since all disk i/o is performed on the RAM disk on each node, the NFS bandwidth was not a result of the models disk i/o for reading input files and writing output files. In these tests, the model was dynamically linked, and we determined that this contributed to the marginal performance. When we ran the tests again with the model statically linked for the case of 96 processors, the speedup doubled to 23.14 with the same bandwidth consumed, indicating that the static link required half the total bandwidth that the dynamic link did. The user CPU utilization doubled to 20% and the system CPU remained at 20%. In these tests, the model was executed by the client via the `system` function. These results are given in Figure 3 as the `system` data series. The graphs show the speedup, bandwidth, and CPU utilization (combined system and user) for varying numbers of processors for a single generation of the genetic algorithm (5 run average).

The speedup achieved for 96 processors was 24.1% of the ideal speedup. While this was a speedup of 23 from the serial case, it was much less than we expected. The bandwidth graphs in Figure 3 indicate that there was considerable and increasing network bandwidth consumed with increasing number of processors. Although the MPI bandwidth was expected to increase, the significant amount of NFS bandwidth was clearly limiting the scalability of the software.

The considerable NFS bandwidth was a result of the technique used for invocation of the model. The client executed the model via the `system` function, which started a shell process (`bash`), and then started a model process within it. The execution of the shell caused the considerable NFS bandwidth usage. We expect that the use of a lightweight shell, such as `sh`, would reduce, but not eliminate, this overhead. To avoid the shell invocation, we modified the client such that the model was invoked via the `fork` and `exec` functions. The upper-left graph of Figure 3 shows the results of using this technique, indicated by the `fork/exec` data series. The use of the `fork/exec` functions for the execution of the model improved the speedup to near linear with considerably less network bandwidth. The percentage of system and user CPU improved to 17% and 83%, respectively.

Although we achieved a near linear speedup, we realized that the overhead of starting the model process had significant performance impacts due to the num-

ber of times it is executed and the short runtime of a single execution. Due to the extensive demands of our error analyses, even a small improvement in speedup may make previously impractical problems feasible. For this reason, we modified the model source code such that it could be invoked as a subroutine. We compiled the model as a static library, but initially received incorrect results due to the model source codes inclusion of Fortran `DATA` statements that have the semantics of a single initialization each time the data segment is loaded into memory during an execution. Since the model is invoked as a subroutine many times during a single execution of the client, the variables initialized by the `DATA` statements must be reinitialized at the start of each invocation of the subroutine. For this reason, we removed all the `DATA` statements (both in the `DATA` block and in the individual subroutines) that initialized variables whose values could change during execution of the model. These variables were moved into common blocks (if they were not present there already) and were initialized in a new subroutine called at the start of the model such that the Fortran 77 semantics were preserved. The library was then linked to the client using standard Fortran-C interoperability techniques. The input and output values to the model are scalars and 1-d arrays which simplified the interoperability. The temporary files used to pass the parameter estimates to the model and retrieve the results from the model were no longer needed. The upper-left graph in Figure 3 shows the results using the subroutine technique, indicated by the `subroutine` data series. The speedup for 96 processors was 106.4 and incurred minimal NFS bandwidth. This super-linear speedup suggests that the overhead of invoking the model as a process (using `fork/exec`) was approximately 30.4%. The percentage of system and user CPU was significantly improved to 9% and 91%, with CPU utilization between 77% and 98% for 96 processors.

The cluster also includes 16 dual-processor AMD Athlon 1.6 GHz nodes connected via gigabit Ethernet. We conducted the same tests on the nodes and received results consistent with those presented above. Future error analyses we wish to perform would require up to a week on our cluster, but if greater resources were used, this runtime could be reduced. We are currently evaluating the performance of our parameter estimation software on Condor at the University of Oklahoma (OU). Condor is a resource scheduler for computing systems composed of dedicated and/or non-dedicated compute nodes. In the case of non-dedicated nodes, the nodes are only utilized when the computer is not in use. The Oklahoma Supercomputing Center for Education and Research (OSCER) has assembled a Condor pool consisting of ca. 200 workstations across campus with plans to increase the number to ca. 1,000.

Condor offers two execution modes for jobs, *standard* and *vanilla*. In both cases, the target program is executed on compute nodes, which in this case are idle workstations across the OU campus. In the standard mode, all system calls, such as file i/o, are remotely executed on the node from which the job was submitted, and checkpointing and migration are supported. In the vanilla mode, the program executes locally on the compute node. To adapt our system to Condor, we only needed to write an alternative client program that submits jobs to Condor rather than to our cluster, requiring no changes to the SciLab script or to the model.

In our initial tests on OU's Condor pool (which used the subroutine technique to execute a single block of 200 EPIC simulations), we found very high variation in the runtime of jobs using the standard mode. Although the network was capable of high effective throughput (200 KB/s) and could achieve good performance (1.4 minutes), in cases where the network provided low throughput (5 KB/s), the runtime was dramatically increased (to 2 hours). This is likely due to the extensive number of individual file i/o operations performed (11,013 operations totaling 66.2 MB read and 7.6 MB written). The use of the vanilla mode yielded much better results, where the runtime was approximately 20.4 seconds. We estimate that with a comparable number of nodes, the Condor system could achieve similar performance to our Athlon cluster. In both cases though, the runtime was highly variable due to the availability of compute nodes and network bandwidth. The average availability of the compute nodes is approximately 89% of the live nodes [personal communication with OSCER]. We plan to extend this work to other high throughput platforms as well, such as BOINC.

5 Conclusions

We have developed a software system for parameter estimation that is composed of a hybrid of software components. Although our cluster is not optimized for the repeated execution of a short-runtime, low-memory program, we were able to achieve efficient use of the cluster. Model-independent, noninvasive techniques in which the model was executed as an independent process could be efficiently implemented through `fork/exec` to achieve near-ideal speedup. This technique though, is inherently limited by the overhead of the process invocation for the model. By invoking the model as a subroutine within a single process, this overhead was eliminated and a 30.4% improvement in speedup over the `fork/exec` approach was achieved for a total speedup of 106.4 with 96 processors.

The paradigm presented is a flexible approach to pa-

parameter estimation suitable for general application. The SciLab implementation of the genetic algorithm provides a familiar and easy to use workbench for tuning the algorithm and experimenting with changes and alternatives. We are currently experimenting with different weightings, parameter ranges, and entropy calculations. The MPI client provides a scalable way to distribute the model executions across the cluster. We plan to utilize the system for the optimization of ecological genomics models in the near future.

Acknowledgments

This work was supported in part by NSF EPSCoR, USDA, and the Provosts Targeted Excellence Program at Kansas State University. The assistance of the Oklahoma University Supercomputing Center is appreciated. Additionally, this material is based in part upon work supported by the National Science Foundation under the award numbers CCR-0082667 and ACS-0092839. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

- [1] D. Abramson, T. Peachey, and A. Lewis. Model optimization and parameter estimation with Nimrod/O. In *Proceedings, International Conference on Computational Science*, pages 720–727, 2006.
- [2] D. P. Anderson and J. McLeod, VII. Local scheduling for volunteer computing. In *the Workshop on Large-Scale, Volatile Desktop Grids held in conjunction with the IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, 2007.
- [3] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski. Toward a common component architecture for high-performance scientific computing. In *Proceedings, 1999 Conference on High Performance Distributed Computing*, 1999.
- [4] E. Cantu-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.
- [5] L. Dagum and R. Menon. OpenMP: An industry-standard API for shared-memory programming. *IEEE Computational Science & Engineering*, 5(1):46–55, 1998.
- [6] J. Doherty. PEST: Model-independent parameter estimation. Technical report, Watermark Numerical Computing, fifth edition, 2004.
- [7] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997.
- [8] A. Golan, G. G. Judge, and D. Miller. *Maximum Entropy Econometrics: Robust Estimation with Limited Data*. John Wiley & Sons, NY, 1996.
- [9] R. L. Graham, G. M. Shipman, B. W. Barrett, R. H. Castain, G. Bosilca, and A. Lumsdaine. Open MPI: A high-performance, heterogeneous MPI. In *Proceedings, Fifth International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, Barcelona, Spain, September 2006.
- [10] K. He, S. Dong, and L. Zheng. Service-oriented grid computation for large-scale parameter estimation in complex environmental modeling. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 741–745, New York, NY, USA, 2006. ACM Press.
- [11] K. He, L. Zheng, S. Dong, L. Tang, J. Wu, and C. Zheng. PGO: A parallel computing platform for global optimization based on genetic algorithm. *Computers & Geosciences*, 33(3):357–366, 2007.
- [12] W. Jin, S. Staggenborg, S. Lauwo, M. Miller, S. Das, D. Andresen, J. Peterson, D. Steward, and S. Welch. Coupling crop and hydrologic models for decision support in the High Plains Aquifer: Preliminary steps. In *Proceedings, 5th International Conference on the Analytic Element Method*, pages 32–43, 2006.
- [13] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.
- [14] E. P. Poeter and M. C. Hill. Documentation of UCODE, a computer code for universal inverse modeling. Technical report, Water-Resources Investigations Reports 98-4080, U.S. Geological Survey, 1998.
- [15] H. Richardson. High Performance Fortran: history, overview and current developments. Technical report, Tech. Rep. TMC-261, Thinking Machines Corporation, 1996.
- [16] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.
- [17] J. Williams and A. Sharpley. EPIC – Erosion/Productivity Impact Calculator: 1. model documentation. Technical report, USDA Technical Bulletin No. 1768, 1989.