**Proc., IBPSA Building Simulation '99,** **LBNL-44097**
**Sept. 13-15, 1999, Kyoto, Japan**

# NUMERICAL PERFORMANCE OF THE SPARK GRAPH-THEORETIC SIMULATION PROGRAM

**Edward F. Sowell**
**California State University**
**Fullerton, CA 92834**

**Philip Haves**
**Building Technologies Department**
**Environmental Energy Technologies Division**
**Lawrence Berkeley National Laboratory**
**University of California**
**Berkeley, CA 94720**

September 13, 1999

**This work was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Building Technology, State and Community Programs, Office of Building Systems of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.**
**Portions of this work were sponsored by the Japan Ministry of Education and the United Kingdom Royal Academy of Engineering Foresight Award Scheme.**

# Numerical Performance of the SPARK Graph-Theoretic Simulation Program

**Edward F. Sowell**
**California State University**
**Fullerton, CA 92834**

**Philip Haves**
**Building Technologies Department**
**Lawrence Berkeley National Laboratory**
**Berkeley, CA 94720**

## ABSTRACT

The Simulation Problem Analysis and Research Kernel (SPARK) uses graph-theoretic techniques to match equations to variables and build computational graphs, yielding solution sequences indicated by needed data flow. Additionally, the problem graph is decomposed into strongly connected components, thus reducing the size of simultaneous equation sets, and small cut sets are determined, thereby reducing the number of iteration variables needed to solve each equation set. The improvement in computational efficiency produced by this graph theoretic preprocessing depends on the nature of the problem. The paper explores the improvement one might expect in practice in three ways. First, two problems chosen to span the range of performance are studied and some of the factors determining the performance are identified and discussed. The problem selected to exhibit a large improvement consists of a set of sparsely coupled non-linear equations. The problem selected to represent the other end of the performance spectrum is a set of equations obtained by discretizing Laplace's equation in two dimensions, e.g. a heat conduction problem. Execution time versus problem size is compared to that obtained with sparse matrix implementations of the same problems. Then, to see if the results of these somewhat contrived limiting cases extend to actual problems in building simulation, a detailed control system model of a six-zone VAV HVAC system is simulated with and without the use of cut set reduction. Execution times are compared between the reduced and non-reduced SPARK models, and with those from an HVACSIM+ model of the same system.

## BACKGROUND

SPARK is a simulation program for non-linear, continuous systems that is currently under development at the Lawrence Berkeley National Laboratory (Buhl, Erdem et al. 1993). The methodology relies upon the mathematical graph for model representation and solution. While ideas from graph theory have been used in connection with equation system solving before (Coates 1959; Harary 1959; Parter 1961; Harary 1962; Tarjan 1972, Steward, 1962; Edwards 1982), in SPARK graph methods are applied directly to the nonlinear equations. The graph, rather than the matrix, is the primary data structure for storing the problem structure and data, and graph algorithms are employed to determine a solution sequence that operates directly on the nonlinear equations. Another distinctive attribute of the approach is that the model equations are stored individually, rather than packaged into modules, and are treated as equations rather than as formulae with assignment (algorithms). Symbolic methods are employed to find explicit inverses of the equations, when possible, to ensure computational efficiency.

In these ways SPARK is unique. However, increasingly, simulation software is employing some of the ideas embodied in SPARK. For example, Klein, in collaboration with F. Alvarado, produced the Engineering Equation Solver, which employs decomposition using sparse matrix methods. This is conceptually the same as the strong component decomposition done in SPARK (Klein 1991). However, reduction within blocks is not done in this software. TRNSYS

has recently been modified to allow “reverse solving” (Fiscal, Thornton et al. 1995). This is a move toward input/output free (non-algorithmic) modeling, another tenet of SPARK. Also in the building context, Tang has applied graph theoretic methods to improve matrix-based solution schemes (Tang 1991; Tang and Clarke 1993).

Although the SPARK methodology is well established there has been relatively little systematic comparisons of solution speed between SPARK and alternative methods available for solving large sets of equations such as arise in building simulation. In order to begin to fill this gap, an experimental research project was undertaken by one of us (Sowell) while on sabbatical at Nagoya University. Two problems sets were tested in this work: (a) a replicated set of four nonlinear equations, and (b) the Laplacian equation, i.e., heat conduction, in a two dimensional grid of various sizes. These two problems, while somewhat removed from mainstream building system simulation, were selected to represent the endpoints in degree to which problems are suited to the methods used in SPARK. To complement findings from these simple problems, the study was extended to include actual building HVAC systems models of considerable complexity. The system selected is one previously studied by Haves (Haves, Norford et al. 1998) using a different building simulation tool, thus providing opportunities for direct comparison.

### NONLINEAR EQUATION EXAMPLE

The first benchmark problem derives from a problem in the SPARK Users’ Manual comprising four highly nonlinear equations:

$$\begin{aligned} x_1 + x_3 + x_2^2 + \sqrt{x_2} &= c_1 \\ x_2 &= x_1 e^{x_1} \\ x_1 x_4 + x_3 x_4 + x_4^3 &= c_2 \\ x_4 &= x_3 e^{-x_3} \end{aligned} \tag{1}$$

SPARK finds a solution to these equations by the calculation sequence:

$$\begin{aligned} x_3 &= 0.1 \\ x_4 &= x_3 e^{-x_3} \\ x_1 &= (c_2 - x_3 x_4 - x_4^3)/x_4 \\ x_2 &= x_1 e^{x_1} \\ x_3 &= c_2 - x_1 - x_2^2 - \sqrt{x_2} \end{aligned} \tag{2}$$

*Iterate on* $x_3$

Using the default SPARK solution process, Newton-Raphson iteration is performed until the difference between two successive values of $x_3$ is less than a specified tolerance. Thus it is seen that reduction of 4:1 is achieved relative to conventional practice of iteration on all unknown variables. With $c_1$=3000 and $c_2$=1 the solution found is $x_4$= 0.288576, $x_1$ = 2.9273, $x_2$ = 54.6738, and $x_3$ = 0.454716.

For the numerical experiments, this set of equations was implemented as a SPARK macro class called *example* which was then instantiated *n/4* times to get a problem of size *n*. Obviously, every instance of *example* is in fact a separately solvable problem. SPARK is able to discern this structural regularity and partition the problem graph into *n/4* strongly connected

components, each with a cut set of size one. Consequently, during the numeric phase of the solution, *n/4* single-variable iterative solutions are carried out. Most conventional, general solvers would instead solve a single equation set of size *n* by iteration on all *n* variables. If Newton-Raphson iteration were used, a linear set with an *n* by *n* coefficient matrix would need to be solved at each iteration.

For comparison purposes this equation set was also solved with two other methods. First, a handcrafted Newton-Raphson nonlinear solver *(nlsolve)* was written specifically for this equation set, with the problem size as an input parameter. In this solver, the four Eqs. (1) were coded in a single function that was called as needed for calculation of the residual functions and the Jacobian. The matrix functions from SPARK were used to numerically calculate the Jacobian and solve the linear set for new estimates of the iteration variables. Second, a sparse Newton-Raphson solver (*spnlsolv*) was written using the sparse LU solve function from the Meschach sparse matrix package (Stewart and Leyk 1994). The interface, function evaluation, Newton-Raphson loop, and output were basically the same as for *nlsolve*, with the only difference being the use of sparse storage date types and sparse matrix solver functions from Meschach. Of course, the same solution tolerance ($1x10^{-6}$) was used in both cases.

Comparative run times with a 333MHz AMD K-6/2 processor are shown in Fig. 1. As would be expected, the experimental results show $O(n^3)$ performance for the full matrix solution. The solver based on the Meschach sparse matrix functions shows much better performance, approximately $O(n^2)$. Also as expected, SPARK is much better than the sparse implementation, showing about $O(n)$.

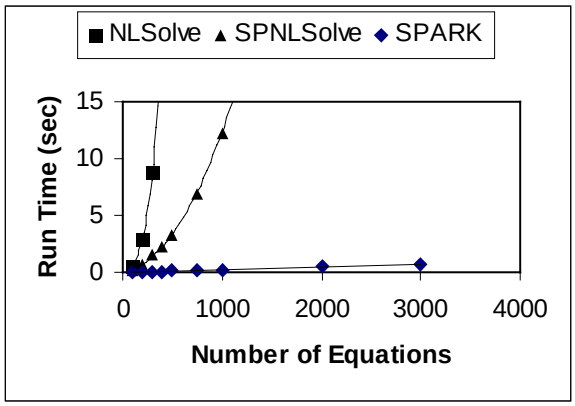


***Figure 1. Solution times for nonlinear benchmark.***

## LAPLACIAN EXAMPLE

The second benchmark problem, purposely chosen to be not well suited to the SPARK methodology, is the 2-dimensional Laplacian equation. This equation models many physical phenomena, including heat transfer in a thin, square plate with uniformly distributed heat source and uniform boundary temperature. The problem is discretized by dividing the square into a uniform grid of specified size. Each cell in the grid is represented by a nodal temperature $T_{i,j}$ and is governed by a heat balance equation

$$q_{si,j} = (T_{i,j} - T_{i,j+1}) + (T_{i,j} - T_{i-1,j}) + (T_{i,j} - T_{i,j-1}) + (T_{i,j} - T_{i+1,j}) \quad (3)$$

where $q_{si,j}$ is the heat source rate per unit surface area. As can be seen, internodal conductance is assumed to be 1.0.

This problem is coded for sparse solution in the Meschach tutorial (Stewart and Leyk 1994). However, for this study that implementation was modified to employ sparse LU factorization, since the use of Cholesky factorization and sparse conjugate gradient iteration in the original code applies only to symmetric positive definite matrices, a condition satisfied by the Laplacian but not often found in general simulation problems.

For comparison, a program was written to generate SPARK problem and input files for the same equation system. The grid size was varied between 3 and 45, yielding equation set sizes between 9 and 2025. Both SPARK and the Meschach-based solver were compiled with the same compiler and optimization options.

In the initial SPARK implementation, each grid node was represented with a SPARK macro object called *node* constructed with atomic *conductor* and *sum* objects from the SPARK HVAC class library (Sowell and Moshier 1995). With this implementation and a grid size of 19 x 19, the SPARK solution time was about 60 times that of the Meschach solver. While a weak SPARK showing was anticipated for this problem, this huge difference was a surprise, calling for further investigation.

The first reason for the long SPARK run times was found to be representation of the node as a macro object. This resulted in seven distinct equations for each node, four of the form $q = u(T_2 - T_1)$ and three of the form $a = b + c$, giving 2530 equations for the grid size of 19 x 19. Although the SPARK graph theoretic algorithms were able to find a cut set of 342, a reduction of 86%, the Meschach implementation was hand-crafted so that there were only 361 equations in the set to be solved. Moreover, the Meschach implementation assumed an inter-nodal conductance of unity, Eq. (3), so no multiplications were needed. Therefore even after graph theoretic reduction brought the Jacobian size down approximately the size seen by Meschach, the SPARK model required many more arithmetic operations in evaluation of each equation. In short, the problems seen by the two solvers were not the same, even though they both represented the same physical problem.

To try to get a more meaningful comparison, both models were changed in several ways. First, the SPARK implementation was revised to more closely approximate the problem as seen by Meschach. A specialized SPARK atomic object class was written to represent the node as a single heat balance equation with an assumed unit conductance, as in the Meschach implementation. With this revision there were only 361 objects in the SPARK model for the 19 x 19 grid, and the SPARK solution times improved considerably.

Then, to see to what extent the presumably more efficient data handling methods in Meschach contribute to its speed advantage, the SPARK solver was modified to optionally use either sparse or non-sparse vector-matrix data structures and functions from Meschach when updating the solution vector. These changes both produced substantial speed-up, with the sparse handling option performing essentially as well as Meschach (See DISCUSSION).

Another difference observed between the two approaches was that because SPARK is a general nonlinear solver it employs Newton-Raphson iteration, requiring numerical calculation of the Jacobian matrix at each iteration. In contrast, the handcrafted Meschach model is aware of the problem linearity and constant coefficients and consequently sets up the conduction matrix only once, directly from the given coefficients. Since this study was concerned principally with solving methods for nonlinear equation systems, it was of interest to see how much of the run time difference was due to extra work in SPARK associated with nonlinear solving. However,

rather than changing SPARK, a second Meschach-based model was developed in which the system of equations was set up for solution as if they were nonlinear. That is, a Jacobian was formed numerically, as in SPARK, and Newton-Raphson iteration was performed to obtain solution. A Meschach sparse solver and supporting vector-matrix routines were used to calculate the solution vector for each iteration. Note that while this approach has the advantages of Meschach's efficient data handling and sparse matrix operations, it does not share SPARK's ability to reduce the Jacobian size.

The results of the various solution methods are brought together in Fig. 2. The three solid curves show SPARK solution times versus total number of equations. The uppermost curve is for solution with the current, standard SPARK. The next lower curve was generated using the modified version of SPARK with the Meschach non-sparse handling of the Jacobian as mentioned above, while the lowest curve results from use of the sparse option. In all three cases, the graph theoretic matching and cutting were coerced with input options so as to get the theoretical minimum cut set size while preserving diagonal dominance of the reduced Jacobian. This is an important qualification and is discussed further below.

The dash-line curve in Fig. 2 is for solution using the Meschach based Newton-Raphson solver described previously. Performance is seen to be significantly better than the standard SPARK, and somewhat better than the modified SPARK using non-sparse methods. However, it is not as good as SPARK using sparse Jacobian handling.

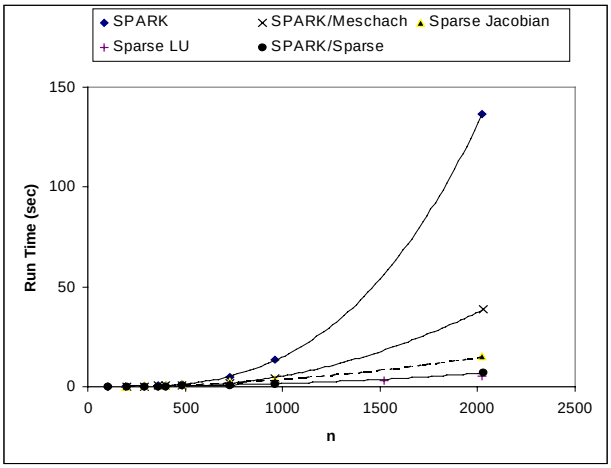


***Figure 2. Solution times for the Laplacian equation.***

The final results in the figure are for the Meschach Tutorial program using sparse LU decomposition. These results overlay almost exactly those for the modified SPARK using sparse Jacobian handling, so a separate trend line is not plotted. However, this agreement is coincidental. Apparently, the reduced Jacobian size in SPARK offers a speed advantage that overcomes SPARK overhead costs such as function calls and numerical Jacobian evaluation, not done in Meschach.[1]

## HVAC BENCHMARKS

Going beyond simple benchmark examples, the numerical methods used in SPARK were also evaluated by modeling an airflow system employing discrete-time controllers. The example used was a typical HVAC airflow network and its associated control loops, a problem involving significant computational burden (Haves, Norford et al. 1998).

A number of steady state component models were implemented as SPARK objects, including variable speed centrifugal fans, flow diverters, flow mixers and control dampers. In modeling air flow, a square law dependence of total pressure drop on flow rate was used above a critical flow

[1] In the current implementation, SPARK makes a call to a C$^{++}$ function for every equation evaluation.

rate and a linear dependence was used below the critical flow rate to avoid known computational problems with air at low flow rates. Dynamic models included flow sensors, pressure sensors, rate limits, discrete-time proportional-plus-integral (PI) controllers and fan control strategies based on PI control.

Figure 3 shows the system that was simulated. The positions of the mixing box dampers determine the proportions of outside and recirculated air that are filtered and cooled before being supplied to the six zones of the building. The positions of the terminal box dampers determine the air flow rates to the corresponding zones. The speed of the supply fan is determined by a PI controller that regulates the static pressure of the air in the supply duct. The speed of the return fan is determined by a PI controller that regulates the difference between the supply airflow rate and the return air flow rate. For the purposes of the benchmark tests, the various damper openings were treated as boundary conditions. In the airflow network used to model the duct system there were 28 flow rate variables and 30 pressure variables, three of which were boundary variables.

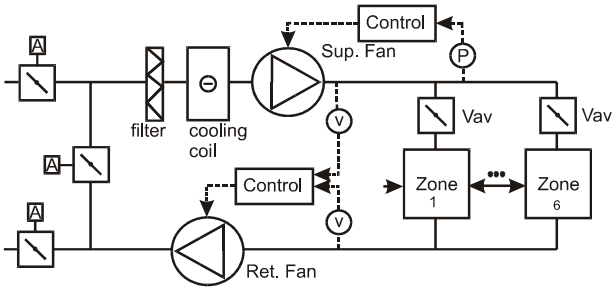


***Figure 3 HVAC system.***

In order to assess the benefits of using SPARK methods, a base case and two reference cases were constructed. The base case was modeled with SPARK in the normal manner, allowing the graph theoretic techniques to perform reduction of the problem graph. The two reference cases were:

- The system modeled using the HVACSIM+ program (Park, Clark et al. 1985), as in the previous work (Haves, Norford et al. 1998).
- The system modeled using SPARK, but inhibiting the normal problem reduction techniques.

The use of the two reference cases enables the benefits of the graph theoretic techniques to be separated from the effects of program architecture.
For all three cases, the simulation carried out was for a period following set point changes at each controller, causing progressive closing of the VAV terminal boxes.

In addition to these comparisons directed at assessment of the importance of reduction, we did a side study to determine whether “breaking” of control loops offers computational advantage. The interest in this derives both from the needs of proper models of discrete time sample-and-hold controllers, and for introduction of artificial delays as a computational device to speed solution.

Comparisons between HVACSIM+ and SPARK are shown in Table 1.

**Table 1** Comparison for HVACSIM+ and SPARK

| | Time (s) | | Iteration Variables | |
|---|---|---|---|---|
| **Control loops** | **HVACSIM+** | **SPARK** | **HVACSIM+** | **SPARK** |
| Intact | 1135 | 48.8 | 62 | 15 |

| | | | | |
|---|---|---|---|---|
| Broken | 785 | 52.7 | 55 | 15 |

In the first comparison, Control loops Intact, the flow network equations and the controller equations are solved simultaneously. The main result is that SPARK is 15 - 20 times faster than HVACSIM+. The obvious reason for the speedup is that SPARK achieves a 4:1 reduction in the number of variables in the iteration vector.

In the second comparison, Control loops Broken, the set of simultaneous equations representing the airflow network and those representing the control system are solved sequentially. This corresponds to breaking the algebraic loops, such as by introduction of a sample-and-hold in the controller, or an artificial delay. Whereas a significant benefit was gained from breaking the control loops when using HVACSIM+, there was no such benefit when using SPARK. The reason for this, as discussed in another paper (Haves and Sowell 1998), is that SPARK finds fan discharge pressures of the supply and return fans to be good choices for break variables, so computation loops are broken regardless.

In order to determine how much of the SPARK advantage can be attributed to the problem reduction, these techniques were disabled, producing the results shown in Table 2 for the Intact loops case.

**Table 2** Effect of SPARK reduction

| | **HVACSIM+** | **SPARK unreduced** | **SPARK reduced** |
|---|---|---|---|
| No. of Eqns. | 62 | 62 | 15 |
| Exec. time (s) | 1135 | 637 | 48.8 |

These results show that the effect of the problem reduction techniques in SPARK is to speed the benchmark problem up by a factor of 13. This is approximately what would be expected from the reduction in the number of equations.

### DISCUSSION

The above results confirm that the SPARK methodology offers significant reduction in solution times relative to conventional matrix methods in the solution of certain kinds of nonlinear equation systems. This is borne out most dramatically by the contrived nonlinear benchmark problem, but is also quite clear from the HVAC control application. However, in the case of the Laplacian example we observe that without some user intervention, SPARK has difficulties competing with sparse methods. Understanding why this occurs is important in order to guide improvement to the SPARK methods, or to properly delineate the class of problems amenable to SPARK methods.

To understand the observed differences in run times it is important to note that at the heart of the Newton-Raphson nonlinear solution process is the solution of a linear problem. That is, during each iteration the solution vector must be updated by solution of the equations

$$J\delta = f(\bar{x}^k) \qquad (4)$$
$$\bar{x}^{k+1} = \bar{x}^k + \delta$$

where $\bar{x}$ is the solution vector of size *n*, $f$ is the vector of functions being solved, $\delta$ is the correction vector, and *J* is the Jacobian. Now, since *J* is *n* by *n*, calculation of its elements is $O(n^2)$, whether done numerically by finite difference (the usual case), or from derivative formulas. Moreover, solution of linear systems is in general an $O(n^3)$ process. Since evaluation of the functions *f* is only of $O(n)$, evaluation of the Jacobian and solving the linear set are the overriding factors in determining run time. Consequently, anything that can be done to reduce the size of the Jacobian has a powerful effect, especially for large problem size.

SPARK gains its advantage over conventional methods by reducing the Jacobian size. It does this in two, separate ways: *decomposition* and *cut set reduction*. Decomposition is possible when the equation set is, in reality, a sequence of separately solvable problems. SPARK is able to detect this property automatically and carry out the decomposition without intervention. For example, the nonlinear benchmark problem with 100 equations and variables is decomposed into 25 subproblems (or, in graph terms, strongly connected components) each of size 4. This alone would reduce the run time from $O(100^3)$ to $25xO(4^3)$, i.e., a factor of 625. Cut set reduction refers to reducing the sizes of the Jacobians of the subproblems. This is done by an algorithm that finds a set of nodes in the problem graph that breaks all cycles, called a cut set. In practical terms, the cut set variables form the iteration vector for the Newton-Raphson process. Again looking at the nonlinear benchmark problem, a cut set of size one was discovered in each component. Thus the 25 Jacobians are all *1x1*, so the overall theoretical run time reduction is by a factor of 40,000. Of course, this efficiency gain is only partially realized due to overhead associated with the SPARK implementation, but this analysis clearly explains the observed excellent performance for this example.

A similar analysis shows why SPARK has difficulties with the Laplacian example. In this case the problem graph, Fig. 4, is more complex, with each node biconnected to four neighbors.

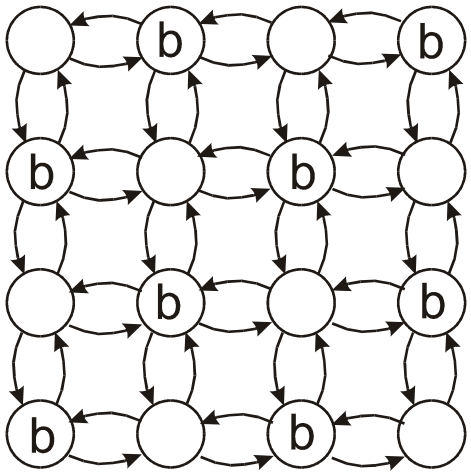


***Figure 4. Laplacian graph.***

One consequence of this high degree of interconnectedness is that the problem does not decompose, so that it has to be solved as a single strongly connected component. Another is that a small cut set is hard to find. The normal SPARK cut set algorithm works on the principle of contraction, in which nodes with single incoming or outgoing edges are bypassed and removed, thereby producing progressively simpler graphs from which the cut set can be deduced. However, there are no such nodes in this graph, so the algorithm must revert to arbitrary removal of nodes into the cut set (Levy and Low 1988). In many problems, arbitrary

removal results in further opportunities for contraction. Such is not the case here, so the algorithm continues to do arbitrary removal, arriving at a relatively large cut set. For example, in the 45x45 grid case (2025 nodes) the discovered cut set is 1894. This is a reduction in Jacobian size of only 5%, hardly enough to overcome overhead costs. Indeed, with this cut set the SPARK run time was nearly 7 times that shown in Fig. 2.

However, it is not difficult to see that a much smaller cut set is possible for the Laplacian. Suppose that for *odd* rows in the grid we mark with **b** (for break) every *even* column node, and apply the reverse policy in *even* rows. This creates a checkerboard pattern on the grid in which every marked node is surrounded by unmarked ones, as shown in Fig. 4. Clearly, the marked nodes form a cut set, since every unmarked node can be calculated given temperature values at the marked ones. This policy can be implemented in a SPARK model using the *break_level* keyword, coercing he algorithm to choose the wanted breaks. When this is done, the cut set size is *n/2,* producing results shown in Fig. 2. In a future version of SPARK it may be possible to improve the matching and cutting algorithms to detect regularities in the problem graph so as to automatically arrive at smaller cut sets in problems of this nature.

While SPARK seeks solution efficiency through graph theoretic reduction, sparse solvers seek it by taking advantage of sparsity in the Jacobian. The first goal in a sparse implementation is to reduce memory usage by storing only nonzero elements in matrices. Secondly, special functions are used to carry out operations such as vector-matrix multiplications with operations performed only on nonzero elements. The Meschach package is very effective in this regard, as evidenced by its performance on the Laplacian problem here. Indeed, the solutions times, shown in Fig. 2, are not only (slightly) smaller than the best SPARK performance, but also are of $O(n^2)$. The reason, of course, is that regardless of the size of the matrix, there are only 5 nonzero entries in each row, and consequently only 5 multiplications and 4 additions in evaluation of each row-vector product. That is, the per-row operations are constant rather than $O(n)$. More advanced sparse implementations go beyond memory saving and efficient vector-matrix operations. For example, there are algorithms that, if possible, reorganize the matrix into block-diagonal form, allowing a partitioned solution that is somewhat equivalent to the strong component decomposition done in SPARK (Tarjan 1972; Klein 1991). The Meschach package currently does not have this feature, as evidenced by its rather poor performance in our nonlinear benchmark example.

An important outcome of this study is the importance of employing sparse methods within SPARK. This is because in problems like the Laplacian the Jacobian can still be quite sparse, even after reduction. In the 45x45 grid, only 1% of the 1012x1012 Jacobian cells are nonzero. This explains the dramatic reduction in SPARK run time in Fig. 2 for the sparse Jacobian modification. Work now underway will provide a sparse solution option in SPARK. This will be selectable on a component by component basis.

The HVAC simulation benchmarks also provide insights into the effectiveness of SPARK solution methodology. From Table 1 we see that SPARK has a clear advantage over HVACSIM+ in simulation of detailed control models. Table 2 also shows that a good deal of the advantage remains even if reduction is not done, raising questions about what other factors are at play. We are unable to fully answer this question, but some contributing factors might be heavier reliance on preprocessing of the problem in SPARK. That is, the graph theoretic analysis is carried out in a separate setup program, which then generates a $C^{++}$ file for compilation. The output of the setup program is an efficient representation of the problem, with the computation sequence more or less built into the data structures. This saves time that a program like HVACSIM+ has to spend moving data from place to place and doing run-time

branching checks and control transfers. Of course, in a large problem (thousands of equations) there can be a significant computational effort involved in the preprocessing step. For short simulation runs, such as the benchmarks reported herein, the time involved may be comparable with or longer than that required to run the problem. However, the SPARK approach has clear advantages for longer or repeated runs.

## CONCLUSIONS

1. Empirical results confirm that SPARK outperforms sparse matrix methods for solution of problems represented by equation systems that can be decomposed and/or reduced with graph theoretical techniques. Roughly speaking, execution time savings will be $O(mr^3)$ where *r* is the ratio of the largest cut set size to the number of equations in the problem, and *m* is the number of strongly connected components into which the problem partitions.
2. Typical HVAC air flow systems, including associated controls, are among the kinds of problems benefiting from the SPARK solution methodology. The reduction techniques produced close to the maximum reduction in the benchmark HVAC problem, and there are indications that similar reductions can be expected in the broad class of problems involving flow networks and their associated control systems. Reductions in execution time of more than an order of magnitude can be expected relative to full-matrix solvers such as HVACSIM+.
3. Problems characterized by a high degree of interconnectivity, such as energy, mass, or momentum transport in homogenous media, allow limited reduction and therefore are not *prima fascia* candidates for SPARK solution methods. However, by proper coercion of matching and cut set selection, significant execution time reduction can still be achieved. Moreover, since the reduced Jacobian in these problems is still very sparse, conventional sparse matrix methods can be beneficially applied after reduction. When this is done, SPARK can be competitive with sparse solvers for this class of problems.

## REFERENCES


1. Buhl, W. F., A. E. Erdem, et al. “Recent Improvements in SPARK: Strong Component Decomposition, Multivalued Objects, and Graphical Interface,” Building Simulation '93, Adelaide, International Building Performance Simulation Association. 1993.
2. Coates, C. L. “Flow-graph Solutions of Linear Algebraic Equations,” IRE Transactions on Circuit Theory **6**: 170-187. 1959.
3. Edwards, D. W. “Robust Decomposition Techniques for Process Design and Optimization,” Ph.D. Thesis, Chemical Engineering, University of London. 1982.
4. Fiscal, A., J. Thornton, et al. “Developments to the TRNSYS Simulation Program,” Journal of Solar Energy Engineering **123**(5). 1995.
5. Harary, F. “A Graph Theoretic Method for the Complete Reduction of a Matrix with a View Toward Finding its Eigenvalues,” J. Math. Physics **38**: 104-111. 1959.
6. Harary, F. “The determinant of the adjacency matrix of a graph,” Society of Industrial and Applied Mathematics **4**(3): 202-210. 1962.
7. Haves, P., L. K. Norford, et al. “A Standard Simulation Testbed for Evaluation of Control Algorithms & Strategies,” Transactions of the American Society of Heating, Refrigerating, and Air-conditioning Engineers **104**(1). 1998.

8. Haves, P. and E. F. Sowell. “The Application of Problem Reduction Techniques Based on Graph Theory to the Simulation of Nonlinear Continuous Systems,” EuroSim, Birmingham, England, Society For Computer Simulation. 1998.

9. Klein, S. . “Engineering Equation Solver (EES),”. Madison, F-Chart Software. 1991.

10. Levy, H. and D. W. Low. “Contraction Algorithm for Finding Small Cycle Cut Sets,” J. Algorithms **9**: 470-493. 1988.

11. Park, C., D. R. Clark, et al. “An Overview of HVACSIM+, a Dynamic Building/HVAC Control Systems Simulation Program,” Proceedings of the First Building Energy Simulation Conference, Dec. 3-6., Seattle, WA, International Building Performance Simulation Association. 1985.

12. Parter, S. “The Use of Linear Graphs in Gauss Elimination,” Society of Industrial and Applied Mathematics **3**(2): 119-130. 1961.

13. Sowell, E. F. and M. A. Moshier. “HVAC Component Model Libraries for Equation-based Solvers,” Building Simulation '95, Madison, WI, International Building Performance Simulation Association. 1995.

14. Stewart, D. E. and Z. Leyk. “Meschach: Matrix Computation in C,” The Centre for Mathematics and Its Applications, The Australian National University. 1994.

15. Tang, D. “The Generalised System Solution Classes in the EKS Environment,” Building Simulation '91, Nice, International Building Performance Simulation Association. 1991.

16. Tang, D. and J. A. Clarke. “Application of the Object Oriented Programming Paradigm to Building Plant System Modelling,” Building Simulation '93, Adelaide, International Building Performance Simulation Association. 1993.

17. Tarjan, R. E. “Depth first search and linear graph algorithms,” SIAM Journal of Computing **1**: 146-160. 1972.

**NOMENCLATURE**

| | |
|---|---|
| $c_l$ | Scalar constant |
| $\delta$ | Correction in Newton-Raphson iteration |
| $f$ | Vector of functions being solved in Newton-Raphson iteration |
| $J$ | Jacobian matrix in Newton-Raphson iteration |
| LU | Lower/Upper matrix factorization. |
| $n$ | Number of equations and variables |
| $O(f(n))$ | Order of notation. The operation in question is bounded from above by $g(n)$ where n is size of data operated on. |
| PI | Proportional-Integral control scheme |
| $q_{si,j}$ | Heat source rate per unit surface node in discrete Laplacian equation |
| $T_{i,j}$ | Temperature of (i, j ) node in discrete Laplacian equation |
| U | Conductance |

$\bar{x}$ Solution vector of size $n$ in Newton-Raphson iteration

$x_i$ Scalar variable

**ACKNOWLEDGEMENT**

**This work was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Building Technology, State and Community Programs, Office of Building Systems of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.**

**Portions of this work were sponsored by the Japan Ministry of Education and the United Kingdom Royal Academy of Engineering Foresight Award Scheme.**