Ten Reasons to Use Divisible Load Theory

Divisible load theory offers a tractable and realistic approach to scheduling that allows integrated modeling of computation and communication in parallel and distributed computing systems.

Thomas G. Robertazzi
State University of New York, Stony Brook

The increasing prevalence of multiprocessor systems and data-intensive computing has created a need for efficient scheduling of computing loads, especially parallel loads that are divisible among processors and links. During the past decade, *divisible load theory* has emerged as a powerful tool for modeling data-intensive computational problems.

DLT originated from a desire to create intelligent sensor networks, but most recent applications involve parallel and distributed computing. The first published research on divisible load theory appeared in a 1988 doctoral dissertation by James Cheng—now director of venture technology at the Siemens Technology-to-Business Center in Berkeley, California—at the State University of New York, Stony Brook. Cheng originally sought to develop intelligent sensor networks that could make measurements, compute, and communicate, but his thesis committee regarded his early work as too theoretical and required more application-related material before giving their approval.

Cheng's original 1988 paper contains an intuitive proof of the DLT optimality principle, but a formal proof did not appear until five years later, following extensive search runs on an IBM mainframe in 1989.[1] The theory's linearity also gradually became apparent in the early 1990s. Since then, an international group of researchers from several countries has participated in a cooperative effort to develop DLT, generating more than 50 journal papers and two books (www.ece.sunysb.edu).

Like other linear mathematical models such as Markovian queuing theory and electric resistive circuit theory, DLT offers easy computation, a schematic language, and equivalent network element modeling.[2] Because DLT does not recognize precedence relations among data, it assumes that computation and communication loads can be partitioned arbitrarily among numerous processors and links, respectively. While it can incorporate stochastic features, the basic model does not make statistical assumptions, which can be the Achilles' heel of a performance evaluation model.

APPLICATIONS

A typical divisible load scheduling application might involve a credit card company that must process 30 million accounts each month. The company could conceivably send 300,000 records to each of 100 processors, but simply splitting the load equally among processors does not take into account different computer and communication link speeds, the scheduling policy, or the interconnection network. Divisible load theory provides the mathematical machinery to do time-optimal processing.

Similarly, banks, insurance companies, and online services often must process large numbers of customer records for billing, data mining, or targeted direct mail advertising, or to evaluate the profitability of new policies. A midsize cap fund would likewise have to process many complex financial records to make the best investment decisions or evaluate new investment strategies.

Processing digital images is yet another such application. For example, it is physically and eco-

0018-9162/03/$17.00 © 2003 IEEE

Simple Divisible Load Theory Example

Consider a five-processor star network with root processor P_0 processing some load itself while simultaneously distributing the rest of the load to processors P_1 thru P_4. Let α_i be the fraction of load processed by each processor. The system parameters are w_i, the inverse computing speed of the ith processor; z_i, the inverse transmission speed of the ith link; T_{cp}, the computation intensity; and T_{cm}, the communication intensity. Thus, $\alpha_i w_i T_{cp}$ is the time to process the ith load fragment on the ith processor and $\alpha_i z_i T_{cm}$ is the time to transmit the ith load fragment on the ith link.

As Figure A shows, a Gantt-chart-like timing diagram can represent a schedule in which transmission commences simultaneously on all links and computation follows load reception on each processor. For a minimum time solution, all processors must stop computing at the same instant; otherwise load could be transferred from busy to idle processors.

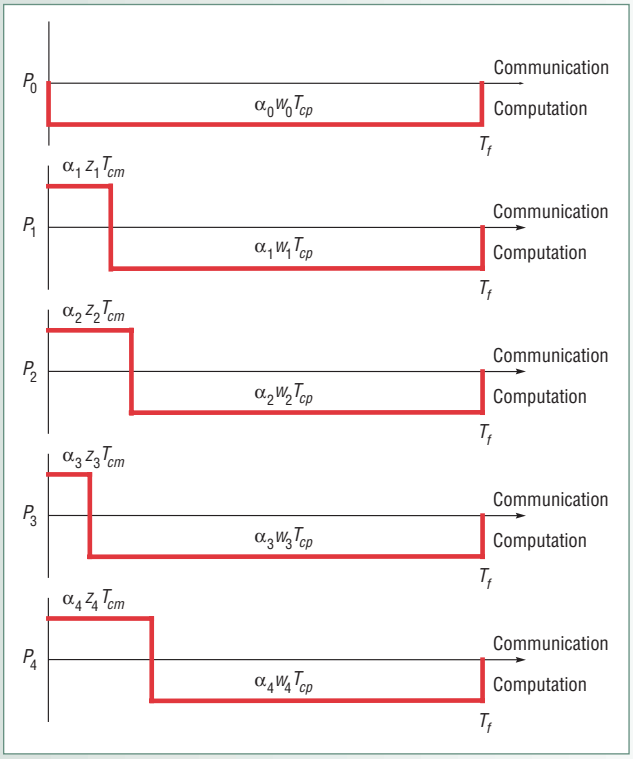

Figure A. Gantt-chart-like timing diagram for star architecture. Transmission commences simultaneously on all links, and computation follows load reception on each processor.

To determine the optimal fragment size for processors 1 through 4, set the equation for the solution time of processor P_0 equal to that for processor P_1, the solution time equation for P_1 equal to that for P_2, and so on. Chaining together the load fragment size solutions results in a complete solution, where m is the number of satellite processors:

$$\alpha_i = \left(\frac{z_{i-1} T_{cm} + w_{i-1} T_{cp}}{z_i T_{cm} + w_i T_{cp}} \right) \alpha_{i-1} \qquad (1)$$

$$= f_{i-1} \alpha_{i-1} = \left(\prod_{j=1}^{i-1} f_j \right) \alpha_1 \quad i = 2, 3 \ldots m$$

For processor P_0:

$$\alpha_0 = \left(\frac{z_1 T_{cm} + w_1 T_{cp}}{w_0 T_{cp}} \right) \alpha_1 = \left(\frac{1}{k_0} \right) \alpha_1 \qquad (2)$$

You can then use normalization ($\alpha_0 + \alpha_1 + \ldots + \alpha_m = 1$) to solve for all the optimal load fractions. If all processor and link speeds are the same, the time to complete a solution is:

$$T_{finish} = \alpha_0 w_0 T_{cp} = \left(\frac{1}{m + 1/k_0} \right) (z T_{cm} + w T_{cp}) \qquad (3)$$

Finally, the linear speedup is:

$$\text{Speedup} = 1 + k_0 m \qquad (4)$$

The same methodology can handle sequential load distribution, sequential load distribution with installments, simultaneous load distribution in which the root does no processing, and simultaneous load distribution in which computation and communication commence at the same time.

nomically impossible for researchers to analyze the millions of images that satellites can quickly generate for particular patterns or features—as required in oil and gas exploration, intelligence gathering, and planetary exploration. Searching millions of digitized fingerprint or facial recognition records for a match presents a similar challenge for law enforcement agencies.

Engineers and scientists at corporations, research labs, and universities also must process large amounts of data for various studies. Even modest engineering experiments can generate copious amounts of data; the collider projects at government-funded physics labs already verge on generating petabytes of data every year. Recent advances in sensor design and implementation, improved data-collection capabilities, and the integration of multiprocessor systems in everything from cars to scientific equipment have created a need for processing and predicting the performance of sensor-generated loads.

TEN REASONS

Given the many potential situations in which a tractable and accurate approach to divisible load scheduling would be useful, I present 10 advantages of using DLT for this purpose.

A tractable model

The *optimality principle*[2] provides the key to divisible load scheduling. Setting up a continuous-variable model and assuming that all processors stop computing at the same instant lets you determine the

optimal amount of total load to assign to each processor or link using a set of linear equations or, as in queuing theory and many other cases, recursive equations. DLT can thus account for heterogeneous computer and link speeds, interconnection topology, and scheduling policy. It can also include fixed delays, such as propagation delay in links. Moreover, as the "Simple Divisible Load Theory Example" sidebar shows, the model can use Gantt-chart-like schematics to easily portray loads with different computation and communication intensities.

DLT's tractable nature contrasts with the traditional indivisible load problem. That is, when you assign atomic jobs or tasks that each must run on a single processor, combinatorial optimization is often NP complete. Precedence relations provide an additional complication. Although not applicable to all computer-scheduling problems, DLT does apply to an important class of such problems in grid computing; signal, sensor, and experimental data processing; and data-intensive and data-parallel computing.

Interconnection topologies

Over the years, researchers have successfully applied divisible load modeling to a wide variety of interconnection topologies, including linear daisy chains,[2,3] trees, buses,[2] hypercubes, and two- and three-dimensional meshes.[4] Figure 1, for example, illustrates a possible load distribution flow originating from a single processor in a 2D mesh network. In addition, asymptotic results developed for infinite-sized networks[2] are useful in sequential load distribution as speedup saturates with the addition of more processors.

The ability to guarantee performance close to that of an infinite-sized network with a small to moderate number of processors therefore provides useful design information. It is worth noting, however, that finding an optimal schedule occurs in the context of a specific interconnection network and scheduling strategy.

Equivalent networks

Like other linear theories, including Markovian queuing theory and resistive electric circuit theory, DLT represents a complex network with an exactly equivalent network element. For some network topologies such as trees, aggregation can be recursive, one subtree at a time.

For example, consider either a pair of adjacent processors and their connecting link in a linear daisy-chain network or a single-level subtree in a multilevel-tree network. You first set the computing speed of a single equivalent processor equal to this subnetwork's speed and then continue aggregating subnetworks of processors, including intermediate equivalent processors, until one processor is left with a computing speed equivalent to the original network. Final expressions for equivalent processor computing speed can be either closed-form or iterative.

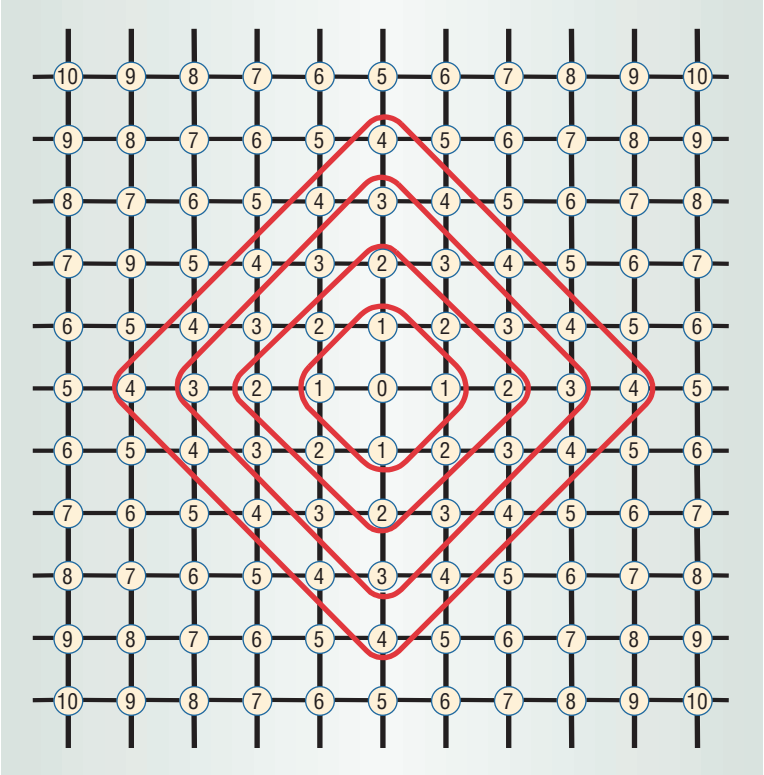

Figure 1. Load distribution flow in a 2D network. The load originates at node 0 and propagates throughout the mesh in a diamond-shaped pattern. (Reprinted with permission from the Poznan University of Technology.)

Installments and sequencing

A number of applied optimization problems arise in divisible load scheduling. For example, instead of a node in a tree sequentially distributing load to its children, improved performance results if load is distributed in installments—some to child 1, child 2, … child M; more to child 1, child 2, … child M; and so on.[5] Performance under sequential multi-installment load distribution strategies tends to saturate as the number of installments increases.

Some sequencing results are surprising. For example, consider a linear daisy-chain network in which all processors and links have the same speed. Under one basic sequential scheduling strategy, if load originates at any interior processor, the same solution time results whether load is first distributed to the left or right parts of the network. Other results are more intuitive. For example, distributing load over a slow link to a relatively fast processor can degrade overall network solution time.[2]

Scalability

Early DLT studies determined that if load is distributed from one node to its children sequentially,

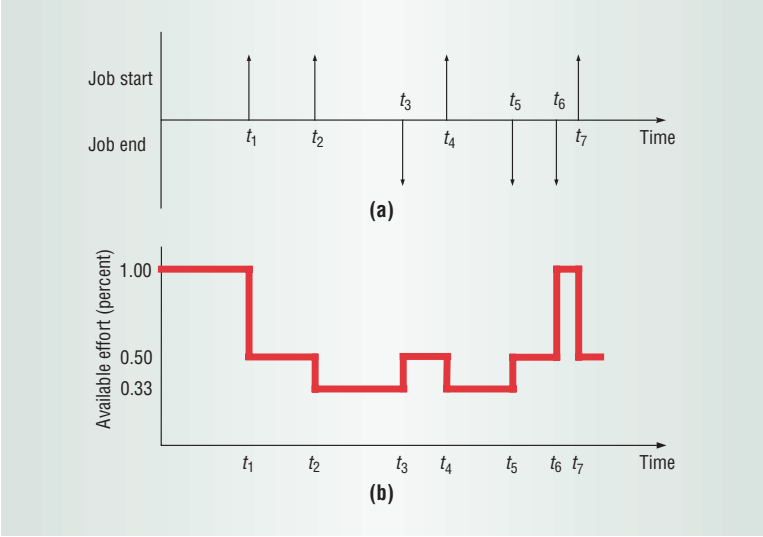

Figure 2. Influence of time-varying background jobs on available processor effort. (a) The upward and downward arrows indicate background jobs commencing and terminating execution, respectively. (b) Normalized CPU effort available for a single divisible job.

as in a tree network, speedup saturates as more nodes are added. If link speed is of the same order as processor speed, optimal sequential load distribution offers a 20 to 40 percent improvement in overall solution time compared to equally dividing the load among processors.[6]

Although simply increasing the number of installments also saturates performance, recent studies indicate that speedup is scalable if a node transmits load simultaneously to all its children—that is, speedup grows linearly in the number of children. As long as a node CPU can load output buffers to all links, performance scales. While there is qualitative support for this scalability concept in parallel processing, DLT allows a quantitative solution.

Metacomputing accounting

A devilish metacomputing problem—distributed computing with payment to computer owners—challenges developers to factor problem size and system parameters into monetary accounting. DLT can incorporate an intuitive linear model for computing and communication costs.[7] Simple to moderately complex heuristic rules can be developed to efficiently assign load in terms of both cost and performance. DLT allows using similar rules for a related problem in parallel processor configuration design, namely how to optimally arrange links and processors with certain characteristics—for example, speed and cost—in a given topology.

Time-varying modeling

The actual effort a computer can devote to a divisible job depends on the status of other background jobs. Ongoing transmissions likewise reduce a link's capacity to transmit part of the job. Figure 2 illustrates the time-varying effort one divisible job receives on a single processor due to background jobs sporadically utilizing CPU effort. Developers can use integral calculus to apply solution-time optimization to divisible loads if they know the start and end times and effort of such background jobs and messaging.[8] With less than perfect knowledge of background processes, stochastic modeling can be combined with deterministic DLT.

Unknown system parameters

It can be difficult to obtain accurate estimates of available processor effort and link capacity, which are key inputs to divisible load scheduling models. Several recently proposed probing strategies[9] send some small fraction of a load to processors across a network of links to estimate currently available processing capacity at nodes and bandwidth on links. Actual implementations must account for the time-varying nature of available processor effort and link capacity as well as processors' release times—the times at which processors become free to accept additional load. Further, load must be distributed on the fastest processors and links. Nevertheless, these probing strategies offer a promising approach to robust divisible load scheduling.

Extending realism

In recent years, researchers have attempted to generalize divisible load scheduling by considering systems with finite buffers,[10] finite job granularity, scheduling with processor release times, and scheduling multiple divisible loads. Other efforts have sought to synthesize deterministic divisible load modeling and stochastic modeling.[6,8] Specialized applications of divisible load scheduling include databases[6,11] and multimedia systems.

Experimental results

Experiments with actual distributed computer systems demonstrate that DLT can be a useful prediction tool, as the "Divisible Load Theory Experimental Work" sidebar illustrates.

FUTURE DIRECTIONS

With investigators scrambling to initiate DLT research in various subareas, there is a need to integrate work to date—for example, to assess time-varying load sharing on hypercubes. In addition, analytical proofs of divisible load optimality exist for only a limited subset of topologies and scheduling policies. While there is no reason to believe that the principle doesn't hold in other environments, rigorous proofs are yet to follow. Beyond these refinements, several potential breakthroughs are on the horizon.

Divisible Load Theory Experimental Work

In 1994, Maciej Drozdowski and colleagues at the Poznan University of Technology in Poland began investigating the accuracy and predictability of divisible load theory. Their research has focused on comparing the real execution time of applications with the model's predictions.

The team conducted experiments on *transputer* systems—simple dedicated platforms that give the user total system control—as well as networks of Sun Microsystems, IBM SP-2, and PC workstations running parallel virtual machines (PVMs), a message passing interface, and Java. The test parallel applications included

- searching for a pattern in a text file,
- compressing a file,
- database join operations, and
- graph coloring using a genetic search metaheuristic.

The difference between predicted values and experimental data obtained using the transputer systems was about 1 percent, due to the systems' simplicity. With workstation networks, however, the difference ranged from 1 percent to as much as 40 percent. The best accuracy occurred with long computations and transfers of large volumes of data. In such cases, the linear part of the computation and communication times dominated.

As computer and communication speeds have increased over the years, the minimum size of divisible jobs that DLT can accurately predict has increased commensurately. For relatively small amounts of divisible data, other phenomena come into play, including

- operating system services' nondeterministic execution time,
- dependence of computation and communication speeds on load size, and
- possible nonuniformity of the load.

Figure B illustrates the difference between modeled predictions and measured values for a relational database join operation on six 133-MHz PCs in a 1999 experiment. The FIFO (first in, first out) graph indicates results returned from processors to the originating processor in the same order that the load was received; the LIFO (last in, first out) graph shows results returned in the opposite order. The model's accuracy increases with job size.

Since 1998, Bharadwaj Veeravalli and colleagues at the National University of Singapore have been implementing scheduling algorithms proposed in the DLT literature to real-life situations that qualify as parallel computations, such as

- low-level processing of images for edge-detection applications,
- large-scale matrix-vector product computations, and
- processing electromagnetic field-strength computations for CAD applications.

For edge-detection applications, the researchers tested the performance of DLT-recommended load distribution strategies on both Hewlett-Packard workstation clusters running PVMs and PC clusters comprised of high-speed Pentium series machines. They processed images ranging from 512 × 512 to 3,000 × 3,000 pixels, which are typical of satellite pictures, under several resource constraints.

Veeravalli's team also implemented a 200 × 100,000 matrix, which is typical in designing industrial microwave ovens and conducting finite-element methods for large-scale mechanical engineering applications, on a PC cluster. They developed a distributed software architecture that carries out the load distribution on a bus network.

Finally, the researchers developed parallelization strategies for SGI machines to compute electromagnetic field strengths around a given circuit layout. This lets CAD designers tune the layout per interference levels between any pair of copper strips in a design.

Overall, Veeravalli and colleagues have reported the difference between DLT predictions and experiment results to be from 5 to 10 percent.

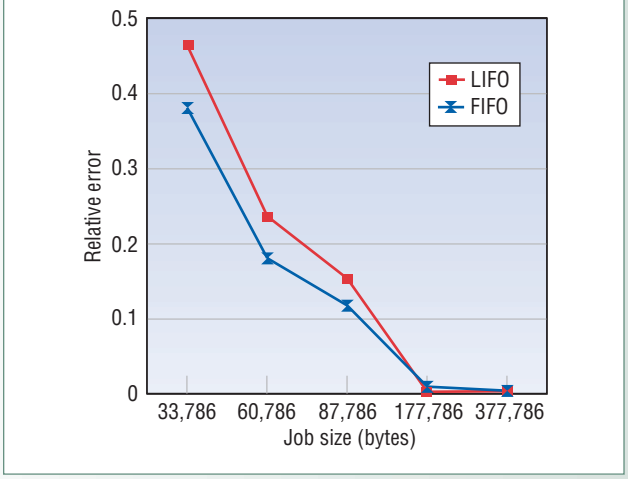

Figure B. Difference between modeled predictions and measured values for a join-type relational database operation. Divisible load theory's accuracy increases with job size. (Reprinted with permission from: M. Drozdowski and P. Wolniewicz, "Experiments with Scheduling Divisible Tasks in Clusters of Workstations," LNCS 1900, Springer Verlag, 2000, p. 318.)

Operating systems and languages

Integrating divisible load scheduling theory with operating systems or data-parallel languages such as high-performance Fortran and high-performance C++ would be a notable accomplishment. Research toward this goal includes pro- cessing divisible and indivisible jobs on the same machine, work on divisible jobs, and multiprogramming and time-varying system analysis.

Monetary cost optimization

This novel application area involves optimizing multiple criteria by minimizing the monetary cost of processing and communicating a load while min-

imizing solution time or maximizing speedup. Potential DLT applications include software brokers that distribute load to vendors doing divisible load as well as other types of processing.

Measurement integration

Researchers have thus far studied the integration of computation and communication in some detail but have largely ignored the measurement aspect. They usually assume that the measurement load is available at an originating processor at certain instants, which in some cases developers model with a Poisson process. We need more elaborate yet realistic models of measurement arrivals and timing and their coordination with computation and communication.

Queuing and divisible model integration

Some efforts have been made to use divisible load theory and queuing theory jointly in modeling,[6,8] but a tightly coupled and tractable integration of the methodologies would be more useful.

The future looks promising for technologies and applications that use divisible load theory. In part, this stems from the flexible analytic structure underlying DLT and its successful cousins, queuing and circuit theory. The breadth of DLT's applications provides another reason for its potential widespread utility. The increasing ubiquity of sensor-generated data, multiprocessor systems, and data-intensive computing create a need for efficient scheduling that should drive further work on theory, applications, and software. ■

Acknowledgments

Maciej Drozdowski, Debasish Ghose, Hyoung Joong Kim, and Bharadwaj Veeravalli provided useful comments on an early draft of this article. I also thank Maciej Drozdowski and Bharadwaj Veeravalli for supplying material on their experimental work, and Jui-Tsun Hung for creating some of the illustrations. The National Science Foundation supports this research under grant CCR-9912331.

References

1. T.G. Robertazzi, "Processor Equivalence for Daisy Chain Load Sharing Processors," *IEEE Trans. Aerospace and Electronic Systems,* vol. 29, no. 4, 1993, pp. 1216-1221.
2. V. Bharadwaj et al., *Scheduling Divisible Loads in Parallel and Distributed Systems,* IEEE CS Press, 1996.
3. Y-C. Cheng and T.G. Robertazzi, "Distributed Computation with Communication Delays," *IEEE Trans. Aerospace and Electronic Systems,* vol. 24, no.6, 1988, pp. 700-712.
4. M. Drozdowski and W. Glazek, "Scheduling Divisible Loads in a Three-Dimensional Mesh of Processors," *Parallel Computing,* vol. 25, no. 4, 1999, pp. 381-404.
5. V. Bharadwaj, D. Ghose, and V. Mani, "Multi-Installment Load Distribution in Tree Networks with Delays," *IEEE Trans. Aerospace and Electronic Systems,* vol. 31, no. 2, 1995, pp. 555-567.
6. K. Ko, *Scheduling Data-Intensive Parallel Processing in Distributed and Networked Environments,* doctoral dissertation, Dept. Electrical and Computer Engineering, State University of New York, Stony Brook, 2000.
7. J. Sohn, T.G. Robertazzi, and S. Luryi, "Optimizing Computing Costs Using Divisible Load Analysis," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 3, 1998, pp. 225-234.
8. J. Sohn and T.G. Robertazzi, "Optimal Time-Varying Load Sharing for Divisible Loads," *IEEE Trans. Aerospace and Electronic Systems,* vol. 34, no. 3, 1998, pp. 907-922.
9. D. Ghose, "A Feedback Strategy for Load Allocation in Workstation Clusters with Unknown Network Resource Capabilities Using the DLT Paradigm," *Proc. 2002 Int'l Conf. Parallel and Distributed Processing Techniques and Applications* (PDPTA 02), vol. 1, CSREA Press, 2002, pp. 425-428.
10. X. Li, V. Bharadwaj, and C-C. Ko, "Divisible Load Scheduling on Single-Level Tree Networks with Buffer Constraints," *IEEE Trans. Aerospace and Electronic Systems,* vol. 36, no. 4, 2000, pp. 1298-1308.
11. M. Drozdowski and P. Wolniewicz, "Experiments with Scheduling Divisible Tasks in Clusters of Workstations," A. Bode et al., eds., *Proc. EURO-Par 2000,* LNCS 1900, Springer Verlag, 2000, pp. 311-319.

Thomas G. Robertazzi *is a professor in the Department of Electrical and Computer Engineering at the State University of New York, Stony Brook. His research interests include parallel processor scheduling, grid computing, networking, and performance evaluation. Robertazzi received a PhD in electrical engineering and computer science from Princeton University. He is a senior member of the IEEE and a member of the IEEE Computer Society. Contact him at tom@ece.sunysb.edu.*