Distribution Category:
Mathematics and
Computer Science (UC-405)

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, IL 60439

ANL-93/40

Using a Transfer Function to Describe the Load-Balancing Problem

by

Andrew J. Conley

Mathematics and Computer Science Division

November 1993

This work was supported by the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38, and by the NSF under Cooperative Agreement No. CCR-9120008.

Contents

Abstract 1

1 Introduction 1

2 Definitions 1

3 The Model 3

4 Consequences of the Model 3

5 Implications 4

6 Conclusion 5

References 5

Using a Transfer Function to Describe the Load-Balancing Problem

by

Andrew J. Conley

Abstract

The dynamic load-balancing problem for mesh-connected parallel computers can be clearly described by introducing a function that identifies how much work is to be transmitted between neighboring processors. This function is a solution to an elliptic problem for which a wealth of knowledge exists. The nonuniqueness of the solution to the load-balancing problem is made explicit.

1 Introduction

The dynamic load-balancing problem for mesh-connected parallel processors can be clearly described by an analogy with problems in vector calculus. Analysis of the equations allows an analytic solution to be found. The analysis also shows that the solution of the load-balancing problem is not unique. The lack of uniqueness is the separate problem of problem partitioning.

The crucial idea of this paper is in the introduction of a transfer function that represents the transfer of work from one node to another. This transfer function can be computed with direct methods or iterative schemes. The model of load balancing that I present includes the iterative schemes of Cybenko [2] and others as a special case. Most important, this paper connects the study of load-balancing with a wealth of information and experience in applied mathematics.

2 Definitions

I assume the processors are laid out on a three-dimensional grid. (The results can be extended easily to one- and two-dimensional grids.) I label the nodes by their distance from the edge processors. For example, a processor in the left front bottom corner is labeled $(0,0,0)$. Processor (l,m,n) is l processors from the left edge, m processors from the front, and n processors from the bottom. The coordinate directions, (x,y,z), are directions of increasing l, m, and n, respectively (see Figure 1).

Now it is easy to define the following:

- Let $L_{l,m,n}$ be the load at node (l,m,n).
- Let $T^x_{l,m,n}, T^y_{l,m,n}, T^z_{l,m,n}$ be the loads transferred to node (l,m,n) from nodes $(l-1,m,n), (l,m-1,n), (l,m,n-1)$, respectively. When $T_{l,m,n}$ refers to transfer of load outside the boundary of the machine, it is defined to be zero (i.e., $T^x_{0,m,n} = 0$).

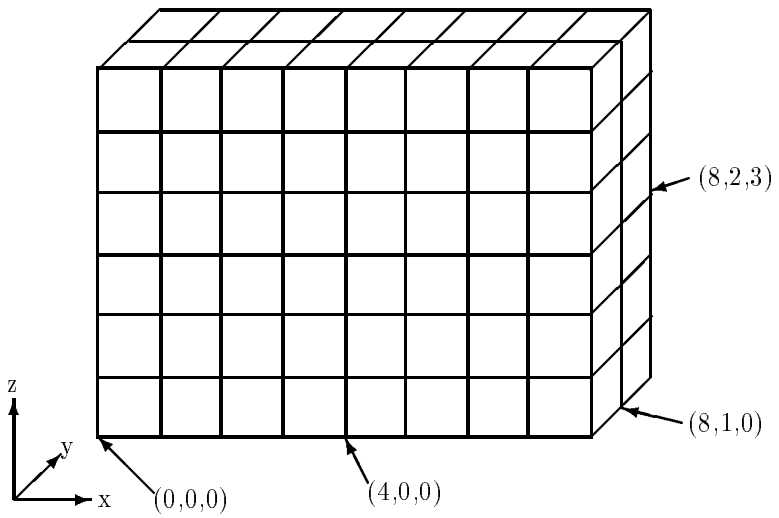

Figure 1: A three-dimensional mesh-connected parallel computer. Several computational nodes are labeled. The load-balancing solution finds the transfers of load between neighboring processors that will make the load on every processor the same.

- Let $L^b_{l,m,n}$ be the load at node (l, m, n) after the load has been balanced.
- Define the gradient of the load at each node as

$$\left[\frac{\partial L}{\partial x}\right]_{l,m,n} = L_{l+1,m,n} - L_{l,m,n}$$

$$\left[\frac{\partial L}{\partial y}\right]_{l,m,n} = L_{l,m+1,n} - L_{l,m,n}$$

$$\left[\frac{\partial L}{\partial z}\right]_{l,m,n} = L_{l,m,n+1} - L_{l,m,n}$$

$$\left[\vec{\nabla} L\right]_{l,m,n} = \begin{pmatrix} \left[\frac{\partial L}{\partial x}\right]_{l,m,n} \\ \left[\frac{\partial L}{\partial y}\right]_{l,m,n} \\ \left[\frac{\partial L}{\partial z}\right]_{l,m,n} \end{pmatrix}.$$

  $\frac{\partial L}{\partial x}$ is defined to be zero at the rightmost nodes. $\frac{\partial L}{\partial y}$ is defined to be zero at the backmost nodes. $\frac{\partial L}{\partial z}$ is defined to be zero at the topmost nodes.

- Similarly, define the flux into a given node as

$$\left[\vec{\nabla} \cdot \vec{T}\right]_{l,m,n} = (T^x_{l+1,m,n} - T^x_{l,m,n}) + (T^y_{l,m+1,n} - T^y_{l,m,n}) + (T^z_{l,m,n+1} - T^z_{l,m,n}).$$

- Define $\vec{\nabla} \times \vec{T}$ as

$$\left[\vec{\nabla} \times \vec{T}\right]_{l,m,n} = \begin{pmatrix} (T^z_{l,m+1,n+1} - T^z_{l,m,n+1}) - (T^y_{l,m+1,n+1} - T^y_{l,m+1,n}) \\ (T^x_{l+1,m,n+1} - T^x_{l,m,n+1}) - (T^z_{l+1,m,n+1} - T^z_{l+1,m,n}) \\ (T^y_{l+1,m+1,n} - T^y_{l,m+1,n}) - (T^x_{l+1,m+1,n} - T^x_{l+1,m,n}) \end{pmatrix}.$$

- Define the discrete Laplacian. (Whenever I apply the Laplacian to a vector, I mean the componentwise application of the Laplacian.)

$$\begin{aligned}\left[\nabla^2 f\right]_{l,m,n} &= (f_{l+1,m,n} + f_{l-1,m,n}) + (f_{l,m+1,n} + f_{l,m-1,n}) \\ &\quad +(f_{l,m,n+1} + f_{l,m,n-1}) - 6f_{l,m,n}.\end{aligned}$$

3 The Model

In the following, I drop the subscripts. All the terms in the equations should be interpreted as being true at all points on the mesh at which the equations are defined.

The change in the load $(L^b - L)$ at any particular node equals the flux of work into the node $(\vec{\nabla} \cdot \vec{T})$.

$$L^b - L = \vec{\nabla} \cdot \vec{T}. \tag{1}$$

However, I wish the load L^b to be balanced after the workload has been transferred. This is the same as saying that the difference (of load) between any two nodes is zero, or, using the notation above,

$$\vec{\nabla} L^b = 0. \tag{2}$$

Lastly, I wish the workload not to be transferred around any particular loop of processors (e.g., work transferred from $(0,0,0) \rightarrow (0,0,1) \rightarrow (0,1,1) \rightarrow (0,1,0) \rightarrow (0,0,0)$). Hence, I require that

$$\vec{\nabla} \times \vec{T} = 0. \tag{3}$$

Equation 3 can be substituted with other constraints (so-called gauge conditions), but then Equation (4) in Section 4 is no longer valid.

4 Consequences of the Model

Taking the gradient of Equation (1), I have

$$\begin{aligned}\vec{\nabla}(L^b - L) &= \vec{\nabla}(\vec{\nabla} \cdot \vec{T}) \\ &\Downarrow \\ \vec{\nabla} L^b - \vec{\nabla} L &= \vec{\nabla}(\vec{\nabla} \cdot \vec{T}) \\ &\Downarrow \quad \text{by vector identity for } \vec{\nabla} \times (\vec{\nabla} \times \vec{T}) \\ \vec{\nabla} L^b - \vec{\nabla} L &= -\vec{\nabla} \times (\vec{\nabla} \times \vec{T}) + \nabla^2 \vec{T} \\ &\Downarrow \quad \text{by Equation 3 and by Equation 2} \\ -\vec{\nabla} L &= \nabla^2 \vec{T}. \end{aligned} \tag{4}$$

I refer to Equation (4) as the flux diffusion equation. The boundary conditions require that transfers outside the mesh be zero. This requirement leads to the Poisson equation (4) for the transfer function with Dirichlet (zero) boundary conditions.

5 Implications

The load-balancing problem is one of computing the transfer of load necessary for the load to be balanced. As the flux diffusion equation (4) makes clear, the transfer function ($\vec{T}$) is the solution of an elliptic problem. The many diffusion schemes that have been proposed are iterative schemes for solving equations (1–2); however, they may violate condition (3).

One naive but effective (nondiffusive) load-balancing scheme might be the following. Every processor balances its load with the loads of all the nodes to its left and right (in its line). Then every processor balances its load with the loads of all the nodes to its top and bottom. Finally, every processor balances its load with the loads of all nodes to its front and back. While this scheme does balance the load, it violates condition (3). Condition (3), however, is necessary in problems where the partitioning of the work is an important issue.

The transfer function ($\vec{T}$) solving Equation (4) is not a unique solution to Equations (1) and (2). The condition (3) could be relaxed by choosing any $\vec{A}$ so that

$$\begin{aligned} \nabla^2 \Phi &= L - L^b & (5) \\ \vec{T} &= -\vec{\nabla}\Phi + \vec{\nabla} \times \vec{A}, & (6) \end{aligned}$$

in order to minimize communication costs during the computational algorithm for which the load is being balanced. The nonuniqueness of the transfer function is the partitioning problem.

The derivation of Section 4 has many other implications. By finding a transfer function that describes the local transfer of load, each processor "knows" how much load it must transfer to its neighbors. Only one local exchange of load is necessary. In the case of large load imbalances, the transfer of load from a processor can be greater than the load on the processor. In this situation, a node can wait until it has received enough work to transfer to its neighbors.

If the architecture is such that the computational nodes are connected to the communication network through communication processors (CPs) (where the CP can compute in addition to communicating), then once the CP knows the load at its processor, the CPs can calculate what the transfers should be, without communicating with the computational nodes.

Equation (4) requires only one Poisson solve, namely,

$$\begin{aligned} \nabla^2 \Phi &= L - L^b & (7) \\ \vec{T} &= -\vec{\nabla}\Phi & (8) \\ \vec{n} \cdot \vec{\nabla}\Phi(\partial) &= 0. & (9) \end{aligned}$$

The solution (Φ) can be computed as accurately as one wishes by any number of algorithms since it is a Poisson solve. (Methods include fast Poisson, Jacobi, Gauss-Seidel, multigrid, domain decomposition, and SOR.) The accuracy of the solution determines the accuracy of the balancing. Solving Equations (7)–(9) requires the same work as the inner loop calculations of the algorithm introduced by Heirich and Taylor [1].

Equations (7)–(9) can be solved analytically in terms of the eigenfunctions, $x_{j,k,p} = \cos(\pi jl/(L-1))\cos(\pi km/(M-1))\cos(\pi pn/(N-1))$, of the discrete Laplacian, where the

number of processors on the grid is LMN. The indices (j,k,p) index the eigenfunctions, and the indices (l,m,n) index the processors on the grid. A Greens function formulation of the solution allows for only local transfer of load and no global solves of the Poisson equation. The Greens function formulation does require that the change in load at every processor eventually be sent to every other processor on the grid.

6 Conclusion

By using definitions of measurable quantities on a multiprocessor, the load-balancing problem can be formulated in terms of the familiar Poisson equation. The solution of this equation (for which there are many techniques) is a transfer function that describes exactly how much load must be transferred locally to balance the load.

References

[1] A. Heirich and S. Taylor. A parabolic theory of load balance. *Caltech Computer Science Technical Report*, 1993.

[2] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *J. Parallel Distrib. Comput.*, 7:279–301, 1989.

Distribution for ANL-93/40

Internal:

J. M. Beumer (60)
A. J. Conley (10)
F. Y. Fradin
G. W. Pieper
R. L. Stevens
C. L. Wilkinson
TIS File

External:

DOE-OSTI, for distribution per UC-405 (54)
ANL-E Library (2)
ANL-W Library
Manager, Chicago Operations Office, DOE
Mathematics and Computer Science Division Review Committee:
W. W. Bledsoe, The University of Texas, Austin
B. L. Buzbee, National Center for Atmospheric Research
J. G. Glimm, State University of New York at Stony Brook
M. T. Heath, University of Illinois, Urbana
E. F. Infante, University of Minnesota
D. O'Leary, University of Maryland
R. E. O'Malley, Rensselaer Polytechnic Institute
M. H. Schultz, Yale University
J. Cavallini, Department of Energy - Office of Scientific Computing
F. Howes, Department of Energy - Office of Scientific Computing