Needed: A Theoretical Basis for Heterogenous Parallel Computing

Arnold L. Rosenberg

CMPSCI Technical Report 94-36
April, 1994

Needed: A Theoretical Basis for Heterogeneous Parallel Computing*

Arnold L. Rosenberg
Department of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003, USA

Abstract. We present a rather general notion of *heterogeneous* parallel computing that calls for a different conceptual basis than does the standard, homogeneous view of parallelism. We illustrate several ways in which the heterogeneous world differs from the homogeneous one, and we enumerate a variety of issues that seem ripe for theoretical study. We argue that the world of heterogeneous parallel machines, which is enabled by emerging technologies and fostered by new economic realities, is orders of magnitude more complicated than its precursors. We suggest that succeeding in this complex world will require closer cooperative research bonds between theoreticians and practicioners than we have seen thus far. We intend the ruminations herein to be viewed as a challenge to both theoreticians and practicioners to create a new conceptual view of parallel computing that addresses the issues raised by heterogeneity.

1 The Changing Profile of Parallel Computation

1.1 The Traditional View of Parallel Computers

When parallel computers first became feasible, they were viewed largely as an evolutionary extension of sequential computers, with the major difference that there were cooperating agents working on parts of the computational chore. Both theoreticians and practicioners adopted this worldview, which led to the conceptual model depicted in Figure 1. The key features of this model are *uniformity* and *predictability*.

Uniformity. In this worldview, the processors and memory modules of a parallel computer are identical in all algorithmically significant features. For the processors, such features include instruction repertoire, speed, and datapath width; for the memory modules, they include capacity, word-size, and latency. Additionally, the communication

* This work was supported in part by NSF Grant CCR-92-21785 and in part by a Lady Davis Visiting Professorship at the Department of Computer Science, The Technion, Haifa, Israel.

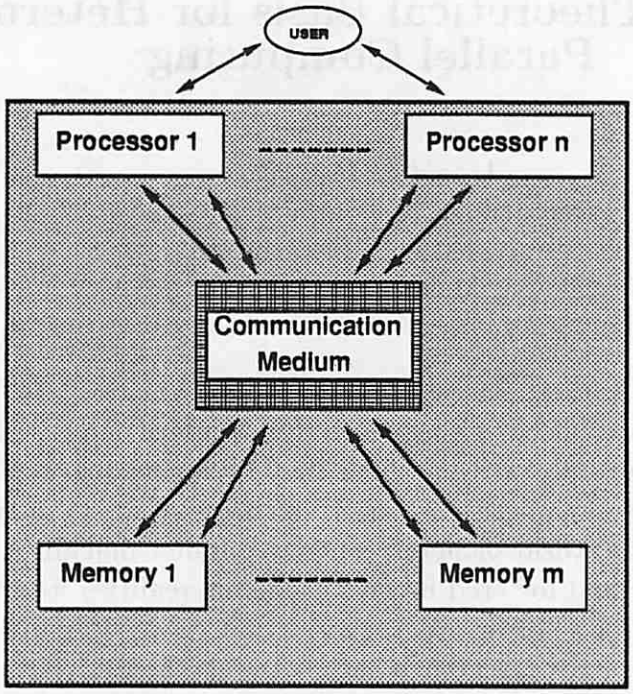

Figure 1: *The traditional view of a parallel computing system.*

medium is "well structured," in the sense of having uniform bandwidth throughout and of having a uniform (and usually exploitable) locality structure. Finally, the entire system submits to a single control regimen (e.g., SIMD or MIMD).

Predictability. In this worldview, processors are 100% available at all times, so that, for instance, computation times can be estimated with known precision.[1] Analyses of algorithms often depend on knowing the values, or at least bounds on the values of performance determinants, such as processor loads, memory latency, and communication bandwidth and latency.

Many of the now-standard parallel algorithmic devices depend in fundamental ways on the assumed uniformity and predictability. The resulting free interchangeability of processors and of memory modules, in terms of capabilities, capacities, and speeds, permit such important algorithmic devices as randomized "compile-time" load balancing and latency hiding via multithreading, as well as all manner of emulation results. Uniformities in algorithmically significant structure further allow the flexible partitioning of machines and the free reallocation of data without costly reformatting. The predictable performances of the processing, memory, and communication subsystems of machines permit

[1] Recent theoretical work on asynchrony and fault tolerance (e.g., [2]) suggest avenues for obviating these assumptions, but most views of parallel computing still cleave to them.

techniques such as "amortized" barrier synchronization and "compile-time" scheduling. It is illuminating to contemplate the extent to which the capabilities and efficiency of "general" models of parallel computers, such as [5, 10], depend on the uniformity and predictability we have been discussing.

1.2 The Emerging View of Parallel Computers

In recent years, increasing numbers of researchers in the world of parallel computing have discovered that the homogeneous world just depicted may be inappropriate in many situations, for a variety of technical and economical reasons. On the technical side, one finds sophisticated application domains, such as computer vision, calling for composite parallel architectures whose constituent subarchitectures are tailored for different aspects of the processing of complex images [6, 11]. On the economical side, the steadily increasing power and moderating costs of uniprocessors, coupled with the continuing high costs of leading-edge multiprocessors has led to parallel computing environments such as: "cooperating"[2] networks of workstations [8, 9, 12]; "stables" of "cooperating" computers of varying speeds and capabilities [3]; and multiprocessors that are shared, via timesharing or partitioning. (An aside: sharing is not usually thought of as a source of heterogeneity, but it does diminish the predictability of an environment, in that different executions of a given program with given data may use dramatically different computational resources.) Such technical and economic factors suggest that one should henceforth view the most general parallel computing system as having the form of Figure 2. The major concomitants of this system are *variety* and *unpredictability*.

Variety. Although most "stables" of parallel computers will likely contain fewer architectural types than appear in Figure 2, they will definitely contain several such types. Moreover, the users of a "stable" will perceive even more types than actually appear, as they receive different portions of the available resources.

Unpredictability. A major challenge for the users of an "aggressively" heterogeneous parallel environment is that the system may keep them guessing about what resources will be available for any given execution of their programs. Most clearly, the various computers in a "stable" will likely vary widely in computing characteristics. However, in a shared environment, even the same computer may exhibit different characteristics at different times. Similar variability will be found in the communication subsystem of the "stable." Since resources may be allocated by the operating system, in a manner influenced by, but not necessarily heeding requests by users, it may be difficult or even

[2] The word "cooperating" indicates that the computers are/may be working on subtasks of the same program, rather than on separate programs.

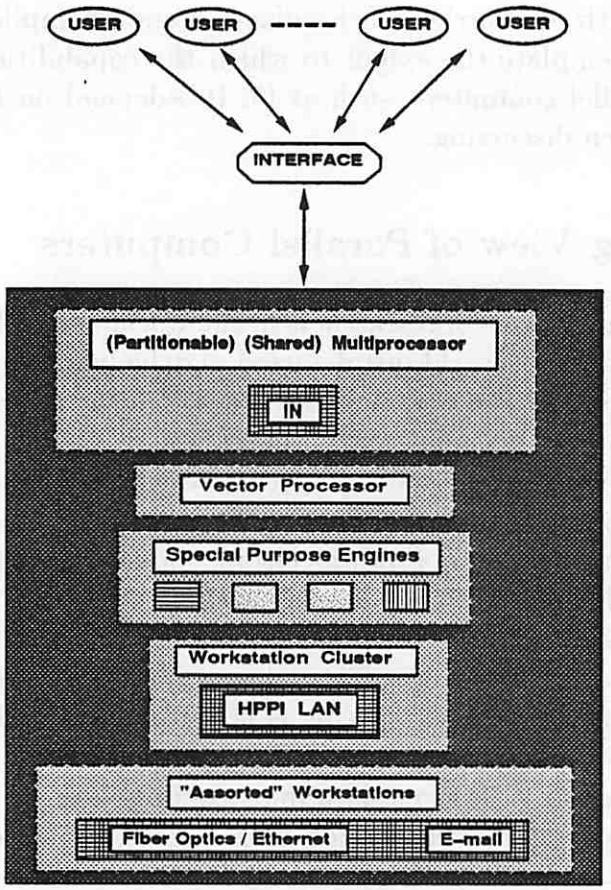

Figure 2: *The new view of a parallel computing system.*

impossible to bound the amount of computational resource that a given program will have access to at any time.

Having sketched a new world of variety and uncertainty, we now try to identify some inviting challenges created by that world.

2 Static Questions Raised by Heterogeneity

2.1 Developing Algorithms in a Varied Environment

Even in the (apparently) simplest heterogeneous environment, there is new algorithmic research needed. To illustrate the problem, consider that existing sophisticated work on well-studied algorithmic problems, such as sorting and matrix multiplication, on a variety

of idealized architectures, such as meshes and hypercubes, does not really prepare one to implement the proposed algorithms on real parallel architectures. The few studies that do proceed from the theory to an implementation suggest that the road from a theoretical study to a well-thought out and well-crafted implementation is not a straightforward one (cf. [1, 7]). How much more sinuous is this road when one has access to a "stable" of varied computers that one can apportion subtasks among?

Reading a source like [4] and pondering the preceding question raises the intriguing subject of *structurally parameterized algorithmics*. Sticking to the realm of algorithm theory for ease of exposition, it appears clear that most algorithmic strategies are robust enough to withstand certain perturbations in the structure of the (ideal) architecture they are targeted for, yet not to withstand others. Algorithms abound that allow one to specify as a simple parameter the number of processors in one's mesh (for instance). It is harder to think of algorithms that enjoy the same (relative) efficiency on processor arrays of different topologies. It is even less likely that there are many algorithms that have equally (relatively) efficient implementations on meshes whose rows and columns are connected by (multireader) buses and on meshes whose connections are all point-to-point. It is hard even to imagine algorithms that are insensitive to the relative speeds of communication and computation. Similarly, it seems likely that developing an algorithmic strategy for a given machine requires careful attention to the detailed characteristics (such as bandwidth and latency) of the machine's communication subsystem. These ruminations suggest the following unformalized, yet compelling questions:

> How much does the structure of the available (sub)machine(s) have to change before an algorithm loses its (relative) efficiency? before one is well advised to select a completely different algorithm?

When one considers suites of interacting algorithms, these questions become even harder. One must now balance the (relative) efficiencies of each constituent algorithm with the desire to execute the entire suite efficiently. As a simple yet extreme example: if one has access to both a mesh and a hypercube, and if all of one's subproblems have algorithms that run significantly faster on the hypercube than on the mesh, then one is faced with an interesting dilemma. Almost certainly, one's optimal strategy will not be to develop a suite of algorithms just for the hypercube, leaving the mesh idle! How, though, does one decide how to partition (and schedule) one's suite of subproblems, given the available "stable" and the available spectrum of algorithmic choices? The decisions become all the harder, obviously, if the available "stable" is as varied as the one depicted in Figure 2.

2.2 Developing Algorithms in an Unpredictable Environment

When one adds unpredictability to the picture, the questions of the preceding section become even harder. The new questions are encapsulated by the following themes.

> How does one specify efficient algorithms when the structure and performance characteristics of the available (sub)machine(s) are unknown until runtime?

Note that the issue of structure arises even in a single-user "stable," if the operating system allocates all work. The issue of performance characteristics arises even when a user does the allocation, if the facilities in the "stable" are shared. As one mixes and matches these possibilities, the opportunities for unpredictability abound.

3 Dynamic Questions Raised by Heterogeneity

The various forms of unpredictability discussed in Section 2 suggest that, in a heterogeneous environment, one may want to defer many algorithmic decisions as long as possible. This requires one to develop algorithms in an environment close to that of *on-line* algorithms (cf. [1]). The dynamic questions that arise can be encapsulated as follows.

> How should work be *allocated physically* and *scheduled temporally* in a computational environment that is heterogeneous? shared? both?

These questions become even more difficult if the environment's vital characteristics (processor and communication speeds, processor availability[3], etc.) may change dynamically, even in the midst of a computation. Such a level of dynamism demands that the system adapt to changes in the environment "automatically," i.e., with no intervention by the user. This last demand notwithstanding, the user must present the system with an algorithmic repertoire that gives it the wherewithal to adapt to change.

The environments we envision demand dynamic allocation and scheduling of work. A client-server model suggests itself, wherein some master server-processor(s) [perhaps many in a shared environment] allocates work dynamically to all other (client) processors in the "stable." The difficulty of implementing such a model in a heterogeneous environment resides in the problem of tailoring a client processor's workload to its computational strengths. Consider, for instance, the following scenario. Say there is just one server, S.

[3] Debilitating faults can obviously render a processor unavailable, but in a heterogeneous environment, so also can a laptop's being disconnected from the network.

At time t, client C is idle and wants work: however, all the work that S has at time t is not particularly well suited for C. Yet, it would be a shame to have C sit idle, while this work has to be done! So, S gives the work to C. Immediately after C has started to do this work, client C'' comes along — and C'' just happens to be the optimal match for the work that S just gave to C. Simultaneously, some more work is generated — which matches perfectly the strengths of C. ... This little unfinished story raises eloquently the issues of task granularity and task migration that challenge any client-server allocation scheme in a heterogeneous environment. It also raises the intriguing question of when it makes sense to double-allocate work, to ensure its timely completion — even when there are no overriding hard deadlines.

Even if one figures out a regimen for matching work to clients, one must decide between having *proactive* and *reactive* servers:

1. Should the server(s) initiate the allocation of work to the clients? This appears to be a somewhat easier platform from which to exploit the individual strengths of the client processors.

2. Should the clients come and request work from the server? This appears to require less system-wide monitoring of computational loads.

Both of these alternatives demand coordination policies, for both clients and servers, to avoid deadlock and starvation, as well as imbalances in workload. Additionally, the first alternative raises the question of how a server should allocate work. One possibly useful metaphor would have a server "auction off" work among the clients. This metaphor requires one to figure out how to run an auction in a highly nonuniform environment, in a way that is computationally unobtrusive. An alternative to auctions would be to have the server(s) monitor client workloads. Models would have to be developed to guarantee a monitoring strategy that probed infrequently enough to be computationally unobtrusive, yet frequently enough to keep client processors adequately busy.

4 Closing Thoughts

It is clear that the world of "aggressive" heterogeneous parallel computing we have been discussing would benefit immeasurably from a system that would yield predictive "back-of-the-envelope" calculations of the efficiency of a proposed allocation and scheduling of work. Neither theoreticians nor practicioners have succeeded in developing such techniques, which could well prove to be indispensable if the world of parallel computation develops in the way suggested in this paper. Indeed, they may turn out to be the real grand challenge of parallel computing.

References

[1] J. Aspnes. O. Waarts. Y. Azar. A. Fiat. S. Plotkin (1993): On-line load balancing with applications to machine scheduling and virtual circuit routing. *25th ACM Symp. on Theory of Computing.*

[2] Y. Aumann. Z.M. Kedem. K.V. Palem. M.O. Rabin (1993): Highly efficient asynchronous execution of large-grained parallel programs. *34th IEEE Symp. on Foundations of Computer Science.*

[3] A. Beguelin. J. Dongarra. A. Geist. R. Manchek. V. Sunderam (1991): Solving computational grand challenges using a network of heterogeneous supercomputers. *5th SIAM Conf. on Parallel Processing for Scientific Computing.* 596-601.

[4] G.E. Blelloch. C.E. Leiserson. B.M. Maggs. C.G. Plaxton. S.J. Smith. M. Zagha (1991): A comparison of sorting algorithms for the Connection Machine CM-2. *3rd ACM Symp. on Parallel Algorithms and Architectures.* 3-16.

[5] D. Culler. R.M. Karp. D. Patterson. A. Sahay. K.E. Schauser. E. Santos. R. Subramonian. T. von Eicken (1993): LogP: towards a realistic model of parallel computation. Typescript. UCal Berkeley.

[6] T.D. deRose. L. Snyder. C. Yang (1987): Near-optimal speedup of graphics algorithms using multigauge parallel computers. *Intl. Conf. on Parallel Processing.* 289-291.

[7] S.L. Johnsson. T. Harris. K.K. Mathur (1989): Matrix multiplication on the Connection Machine. Tech. Rpt. TR-736. Yale Univ.

[8] M. Litzkow. M. Livny. M. Matka (1988): Condor - A hunter of idle workstations. *8th Ann. Intl. Conf. on Distributed Computing Systems.*

[9] D. Nichols (1990): *Multiprocessing in a Network of Workstations.* Ph.D. thesis. CMU.

[10] L.G. Valiant (1990): A bridging model for parallel computation. *C. ACM 33.* 103-111.

[11] C.C. Weems. S.P. Levitan. A.R. Hanson. E.M. Riseman. D.B. Shu. J.G. Nash (1989): The Image Understanding Architecture. *Intl. J. Computer Vision 2.* 251-282.

[12] S.W. White and D.C. Torney (1993): Use of a workstation cluster for the physical mapping of chromosomes. *SIAM NEWS.* March. 1993. 14-17.