Models of Parallel Computation: A Survey and Synthesis

B.M. Maggs [*]
Carnegie-Mellon University
Pittsburgh, Pa 15213

L.R. Matheson
NEC Research Institute
Princeton, NJ 08540

R.E. Tarjan [†]
Princeton University
NEC Research Institute &
Princeton, NJ 08540

Abstract

In the realm of sequential computing the random access machine has successufully provided an underlying model of computation that promoted consistency and coordination among algorithm developers, computer architects and language experts. In the realm of parallel computing, however, there has been no similar success. The need for such a unifying parallel model or set of models is heightened by the greater demand for performance and the greater diversity among machines. Yet the modeling of parallel computing still seems to be mired in controversy and chaos.

This paper is an excerpt from a study which presents broad range of models of parallel computation and the different roles they serve in algorithm, language and machine design. The objective is to better understand which model characteristics are important to each design community in order to elucidate the requirements of a unifying paradigm. As an impetus for discussion, we conclude by suggesting a model of parallel computation which is consistent with a model design philosophy which balances simplicity and descriptivity with prescriptivity.

Space constraints allow only the presentation of the survey of abstract computational models. It is our hope that the introduction provides insights into the rich array of relevant issues in other disciplines, inspiring the interested reader to examine the full paper.

*Research partially supported by NSF National Young Investigator Award and ARPA Grant Nos. F33615-93-1-1330, N00014-91-J-1698, N00014-92-J-1799

†Princeton research partially supported by NSF grant no. CCR-8920505 and Office of Naval Research Contract No. N0014-91-J-1463

1 Introduction

Modeling complex phenomena is as old as science itself, perhaps as old as art itself. Choosing the right characteristics to model and incorporating them simply, elegantly and accurately requires as much artistic creativity as scientific methodology. Even the early Greek mathematicians, who were primarily interested in geometry, dealt with the issues of modeling. Indeed, Pythagoras discovered that the set of rational numbers (ratios of whole numbers) were inadequate to describe the length of a diagonal of a square having sides of length 1 [31]. This made it necessary to augment the rationals to create a more powerful and expressive tool: the real numbers. Since the time of the Greeks, the art and science of modeling has mushroomed. Today models are ubiquitous - controlling large portions of our financial markets, routing our air traffic, explaining the nature of our genetic makeup, tracking our weather, and predicting our overall economic health.

Why Model?

Broadly stated the purpose of modeling is to capture the salient characteristics of phenomena with clarity and the right degree of accuracy to facilitate analysis and prediction. Computer scientists use models in a myriad of ways to help design efficient problem solving tools. These tools include fast algorithms, effective programming environments and powerful execution engines. Usually the alternatives to modeling are resource-intensive. Implementation, for example, is time consuming and provides information which is accurate but too narrowly focused to support general conclusions. Models, on the other hand, can provide a broader view inexpensively, illuminating general patterns and issues, but they suffer from a lack of specific accuracy. Modeling can be used interactively with implementation in a symbiotic process of problem solution which is far more efficient than using either approach separately.

An Objective and a General Framework

In this paper we examine how computer scientists are using models to better solve problems on parallel computers. The goal is to promote discussion of the characteristics and objectives of these models to better determine which ones are essential and how unification might be achieved. To accomplish this we first present a selected survey of models from several of these disciplines. The survey is presented within a simple logical framework, first proposed by Snyder in [30]. This framework allows the wide array of models used in computer science to be viewed somewhat systematically.

The solution to any given task begins with the design of a set of steps (an algorithm) which will realize the computational solution to an abstract problem specification. Problems can be come from many different domains: e.g. mathematical, biological, logical. In each domain translation from problem to computational algorithm requires a model of computation. For example, the development of serial computing has produced, among others, the widely accepted Von Neumann model, expressed elegantly in the random access machine model (RAM) [9]. Such a **computational model**[1] must clearly define an execution engine powerful enough to produce a solution to the relevant class of problems. In addition such a model needs to reflect the salient computing characteristics of practical computing platforms. These objectives, one prescriptive and the other descriptive combine to enable the translation from abstract formulation to an algorithm which gives the desired solution.

An algorithmic specification is then translated into a sequence of machine-independent software instructions. This translation process is facilitated by a **programming model**[2] Formally, a programming model provides a set of rules or relationships that defines the meaning of a set of programming abstractions. These abstractions are manifested in a programming language which is an instantiation of that programming model. A primary objective is to allow reasoning about program meaning and correctness. For example the rules of lambda calculus define the meaning of functions and applications which serve as the basis for the abstractions found in several functional programming languages. Practical programming models often lack such rigor and the definition of a programming model has evolved to imply a set of language constructs that can be used to express an algorithmic concept in a programming language. For example, the imperative-procedural model posits constructs such as arrays, control structures, procedures and recursion and the programming languages Pascal and C are designed within this model, while the constructs Lists, Cons, Apply are encompassed by a functional programming model.[3]

Historically, the primary focus of programming language development has been on expressibility, providing constructs which elegantly, and perhaps provably, translate and preserve algorithmic intentions. But language development also requires the translation of the high level language constructs into machine-dependent executable instructions. This translation process is usually facilitated by not by the use of a progamming model but by the use of a model of the execution engine. These machine models are used to tune the performance of a programming language and typically express the cost of crucial machine operations. Thus, in the realm of parallel computing the demand for performance has begun to realign the focus of language development to include both goals: expressability and performance.

Low level machine models or **architectural models**[4] describe a class of models used for a broad range of design purposes such as language implementation and machine design. These models need to reflect the detailed execution characteristics of actual or envisioned computers. These models cover a wide range: from fairly high level representations on an architectural level to instruction execution models or even detailed component models. These models are primarily used in machine design and the objective of the models is almost exclusively performance. These models are commonly used in parallel computing to compare alternative architectures, often for a given class of problems or alternatively predicting the performance of a specific machine on a set of algorithmic strategies. Performance analysis using these models expresses the design characteristics and concerns of evolving technologies and provides feedback into many aspects of task solution including the design of the machine itself.

[1] Terminology is an immediate, recurrent, labyrinthine problem in any discussion of models. Computational models are alternatively called abstract machine models, cost models and performance models. In this paper, the terms computational model and abstract machine model are used interchangeably and synonymously.

[2] The term programming model persistently defies precise definition.

[3] The astute reader will recognize this as a broad generalization used for illustrative purposes only as many programming languages possess characteristics attributable to both of these programming models.

[4] These are also called cost models, performance models, and hardware models.

While this framework provides a tool to view some of the models used in computer science and their objectives, it also illuminates the need for model consistency. The greater the distance between these classes of models and actual platforms, the more difficult and inefficient each of the design and translation processes becomes and the poorer the results. While the rewards for consistency are fairly apparent, much less apparent is the means of achieving this consistency. To date there seems to be some consensus on the importance of the problem [29] yet there has been no single model or set of models which has achieved widespread accceptance and managed to achieve this consistancy.

Serial vs. Parallel Models of Computation: Why the Chaos?

The history of serial computing has demonstrated that a simple abstract machine model can facilitate consistency by providing a clear and simple rendering of the execution engine. Unfortunately the search for such a unifying parallel paradigm has been far less successful. One reason is the high standard of simplicity instantiated by the RAM. The natural parallel analog, the parallel random access machine (PRAM) model, meets this standard but is by no means as accurate: unmodeled characteristics, most importantly the cost of a non-local memory reference, have a great impact on performance. The RAM was accurate enough to enable the use of asymptotic performance measures. This was in part due to a balance among the unmodeled machine characteristics such the cost of a memory access and an arithmetic instruction. Parallel computers, however, lack this balance, and asymptotic analysis is a far less effective design tool. But the standard of asymptotic analysis set by the RAM has made problematic the broad acceptance of a parallel model complicated by constant factors. The development of a unifying paradigm also requires a somewhat unified and stable technological environment. In parallel computing, not only are there intra-generational variations but there are broad architectural changes between generations. Put simply, the development of a unifying abstract machine model requires unifying several different moving targets.

Descriptive vs. Prescriptive Models

The modest convergence towards a common set of architectural characteristics in the current generation of massively parallel computers provides some stability on which to base an effective computational model. Recent modeling efforts have introduced more complexity of description and constant factors. But while these models tend to more accurately describe actual computers, the current generation of machines has some marked deficiencies which hinder their ability to be efficient platforms for the solution of certain classes of problems.

There is a strong motivation for a model design philosophy which balances the need for descriptive accuracy with the need to provide a medium which prescribes machine characteristics which will promote efficient solution of classes of important problems. An integration of descriptivity and prescriptivity is more effective than either as an exclusive modeling philosophy. Portraying an inefficient computational platform accurately provides distorted design incentives. Rather than assisting the development of clean solution ideas, the model tends to focus resources on clever ways to circumvent or cloak the platform inefficiencies. On the other hand designing effective solutions for the ideal platform with no consideration of realizablity is equally as counterproductive. Optimally, a model should provide clear, productive design incentives while providing strong messages to platform designers about the quality of characteristics required for efficient solution.

Summary

In the realm of parallel computing there is a compelling need for models of parallel computation which facilitate an efficient design process and produce efficient designs. This requires an underlying consistency throughout the models used for different design and implementation tasks. This consistency, though difficult to achieve in a rapidly changing environment, may be best promoted by the development of a powerful abstract machine model. But this model needs to answer to a tall order in balancing simplicity with accuracy, abstraction with practicality, and descriptivity with prescriptivity. To help understand what such a model should look like, the following sections selectively survey existing parallel computing models.

2 Computational Models

The Family of PRAMs

The success of the serial random access machine model spawned a family of parallel random access models in the search for a simple unifying abstraction to guide design. In its simplest form the parallel random access machine (PRAM) model [20] posits a set of P processors, with global shared memory, executing the same program in lockstep. Though there is some variability between PRAM definitions, the standard PRAM is a MIMD computer where each processor can execute its own instruction stream. Every processor can access any memory location in one time step regardless of the memory location. The main difference among PRAM models is in how they deal with read or write memory contention. This is discussed more fully below.

Despite its impressive dissimilarity to practical parallel computers the PRAM can provide useful design information. By ignoring any (and all) costs associated with exploiting parallelism, the incentives in the model lead the designer to expose the maximum possible computational parallelism in a given task. Thus the PRAM provides a measure of the ideal parallel time complexity.

Stalwart PRAM critics do not share this favorable view. Instead they point to economic externality arguments [25]. Without associating a reasonable and practical cost to the use of a resource (computational parallelism), there is an incentive to use the resource abusively. In other words the aspects of practical computing hidden by the PRAM serve only to distort the design process and including some of these aspects early in this process can only increase the potential for efficient solutions.

The critical question to pose to these critics then is: which characteristics should be incorporated and by what means? There have been a profusion of proposed answers to this question. There have been many surveys of the proposed variations on the PRAM, for example ([17, 28]). Presented below is a representative subset that illustrate which practical machine characteristics have been the focus of efforts to improve the PRAM.

1.*Memory Access.* Among basic variants of the PRAM, the most powerful is the CRCW PRAM [20] which allows concurrent reading or writing of any memory location, with some rule to resolve concurrent writes, such as randomization, prioritization, or combining. Concurrent memory access at unit cost is perhaps the most egregious aspect of this model, and many variants have been developed to restrict this idealization. The EREW and CREW PRAMs [20] restrict access to a given memory location to one processor at a time, either for both reads and writes (EREW) or just for writes (CREW). This preserves unit access cost but enforces some notion of serial access. Another PRAM variant which arbitrates memory access is the *Module Parallel Computer* (MPC) [27]. In this model shared global memory is broken into m modules. Only one memory access can occur within each module per time step. The LPRAM (or Local-memory PRAM) proposed in [3] augments the CREW PRAM by associating with each processor an unlimited amount of local private memory. The QRQW PRAM [16] provides an intermediary, a queue, to arbitrate and manage memory accesses, while charging a memory access cost which is a function of the queue length. These arbitration devices all tend to provide the incentive to avoid target memory location contention.

2.*Synchronization.* The standard PRAM posits a rigid execution pattern in which all processors are synchronized by a global clock. Several variants ease this restriction. Examples which allow asynchronous execution with irregular synchronization points include the APRAM [10] and the Asynchronous PRAM [15]. Periodic synchronization between intervals of asynchronous execution is incorporated in the XPRAM [34]. While these models incorporate synchronization, they do not charge an explicit cost. Although the only cost is implicit, (the loss of processor utilization while waiting for other processors to complete) these models still provide an incentive to synchronize only when necessary.

3.*Latency.* As it became clear that the cost of non-local memory accesses has a severe effect on performance in massively parallel computers, several PRAMs were designed to remedy the unit-memory cost idealization. It was suggested that the LPRAM could be aumented by charging a cost of l units to access global memory [3]. An elaboration of this model, the BPRAM [4], augmented this by charging l units for the first message from global memory and a variable cost, b, for each additional memory access in a contiguous block. Thus the BPRAM provides incentives for one level of reference locality and for block transfer, a form of data parallelism. Oddly enough this model was proposed in 1989-90, before the current generation of massively parallel computers existed. It is in this new generation, as opposed to the previous generation, that a very high fixed communication cost is incurred and usually coupled with a low variable cost per byte.

4.*Bandwidth.* Another example of a PRAM variant which assumes two classes of memory and includes a mechanism for assigning a non-unit cost to a remote access is the DRAM [23]. The DRAM is important because it eliminates the paradigm of global shared memory and replaces it with only private distributed memory. While the topology of the communication network is ignored, the DRAM incorporates the notion of limited bandwidth. This model proposes a cost function for a non-local memory access which is based on the maximum possible congestion for a given data partition and execution sequence. While the function is somewhat complicated it attempts to provide scheduling incentives to respect limited access to non-local data. A recent PRAM variant proposed in [36], the PRAM(m) incorporates bandwidth limitations by restricting the size of global shared memory to m memory locations. The model is a CRCW PRAM except that in any given step only m accesses can be serviced.

5.*Primitives.* The examples above include some of the proposed modifications to the PRAM which better reflect the characteristics of actual computers. Many other variants have been developed which enhance the standard PRAM. Some for example include additional unit-cost primitive operations such as scans [7]. Such models can be considered to be designed to be prescriptive, suggesting features, some more realizable than others, which could potentially make a parallel platform efficient for a given class of problems.

Other Distributed Models

The perceived technical infeasibilty of constant-time access to a global address space led to development of many PRAM variants. A fundamental paradigm shift, however, was the introduction of models in which modules of memory are associated with processors. An early example of this type of model is used by Upfal in [32]. Later referred to as the *Distributed Memory Model* (DMM), it posits private memory modules associated with processors in a bounded degree network. Computation and nearest neighbor communication require one time step.

Another more recent example of a distributed memory model is the *Postal Model* [6] deriving its name from an analogy to the US mail system. In this model to accomplish a non-local memory access a processor posts a message into the network and goes about its business (posting other messages) while the first is being delivered. This model is notable for its explicit mechanism and incentive for latency hiding. The model is strictly a model of communication and provides no description of computation. A related model which includes computation but introduces quite a bit more complexity is the *Atomic Model for Message Passing* developed in [24].

Low-Level Models

Many abstract models have been recently developed which incorporate a more detailed view of the machine components and behavior. The objective of these *Low-level models* is often to assess the feasibility and efficiency of a particular machine or component design, sometimes for a particular class of algorithms, or to understand which particular algorithm or implementation may be most efficient on a given machine or component design. For example, in [7] Blelloch et al. develop a detailed model of the CM-2 by Thinking Machines Inc., a fine-grained hypercube-connected massively parallel computer, in order to better understand which sorting algorithms and implementations perform best on this platform. Their model includes a two level memory hierarchy and four primitives which closely correspond to the actual machine primitives. The cost of the primitive operations is approximated using actual machine timings and the analysis considers constant factors. The model incorporates the cost of an arithmetic operation, a nearest neighbor communication, a general communication and a scan operation. A similar analysis is performed using a model developed to reflect the MasPar MP1 computer in [19]. This model, also used to draw conclusions about the efficiency of alternative sorting algorithms, employs a very similar set of parameters.

Hierarchical Memory Models

Because data storage in a computer is accomplished through a variety of different physical units and media and access time is very different among these media, there has been a sustained interest in modeling this phenomenon. In most such models the concept of random access is irretrievably altered. Two early examples of serial hierarchical memory models are the HMM [1] model and the BT [2] model. In the HMM model there are k levels of memory each of which contains 2^k memory locations; access to memory location x takes $f(x)$ time for some function f. The BT model, (the HMM model with block transfer), incorporates the possibility of moving data in large blocks.

While serial models such as these can provide practical design tools for special classes of problems which involve large amounts of data movement, they are not widely used as general design tools. Yet several arguments exist, such as the need for massive amounts of data movement in many current and envisioned parallel applications, to motivate the inclusion of a memory hierarchy in a general purpose model of parallel computation. The parallel versions of these models, the P-HMM and P-BT models, replicate the serial model P times and connect the processors and memory through a network which allows parallel data movement.

Generally, this class of models seeks to provide a more refined and practical reflection of memory access cost by defining a hierarchy with monotonically increasing sizes and access costs for each subsequent level. In these models data movement is a valued resource, and the models provide performance rewards for exploiting both types of data movement parallelism: block transfer and parallel transfer. A perspective on the potential benefits of data movement parallelism can be found in [5] and a survey of parallel hierarchical memory models can be found in [37]. For problems which involve the movement of large amounts of data these models may be particularly effective in producing efficient algorithms. An example of algorithm design using these models can be found in [38].

Network Models

Both classes of models discussed above ignore the possible impacts of the topology of the communication network. Network models of parallel computation reflect a focus of concern in the early generation of parallel computers. These computers tended to be fine-grained, composed of a large number of relatively small processors. Network models generally ascribe some amount of local memory to each processor. The cost of a remote memory access is a function of both the topology and the access pattern. The cost functions tend to be completely variable with no fixed start-up costs for communication. These models provide design incentives for efficient data mappings and communication routings. There are at least as many models as there are proposed network topologies. A survey and analysis with an exhaustive bibliography can be found in [22].

While the standard PRAM presents an uncluttered and appealing design platform, it presents little opportunity for optimization with respect to practical machine attributes. Each of the classes of models discussed subsequently introduces some practical aspect to the user, thereby conferring some responsiblity for optimization. The interested reader is urged to consider the models not included here because of space constraints.

Bridging Models

The notion of a bridging model was effectively captured by Valiant when he described the von Neumann model as "a connecting bridge that enables programs to run efficiently on machines from the diverse and chaotic world of hardware" [33]. In [35] Valiant provides compelling arguments for an abstract parallel machine model which provides a unifying and consistant design paradigm to facilitate portable parallel algorithm design and program translation.

In an early multidisciplinary effort Snyder proposed one possible bridging model, the Candidate Type Architecture model [30]. This model posits a finite number of sequential von Neumann computers executing asynchronously, with a global synchronization mechanism, connected in a network of fixed bounded degree. The model specifies communication cost, but synchronization, achieved through the global controller, is free, and there are no bandwidth constraints. This simple two parameter model (communication cost L and number of processors P), with a two-level memory hierarchy, provides incentives for reference locality. The model does not provide explicit incentives for latency hiding, bandwidth management, or synchronization avoidance. It assumes the opportunity to exploit these optimizations will be recognized by the programmer and provides explicit constructs for these optimizations in an associated programming model.

Valiant's own bulk Synchronous Parallel (BSP) model posits a distributed memory with three parameters. The model provides P processors with local memory, a router, and facilities for periodic global synchronization. Computation can be synchronized at most every l steps and the ratio of local units of computation to the steps required to transmit or receive a message is a parameter g.

These three parameters serve several functions. First, the parameter l reflects the cost of invoking a synchronization operation. It also implies a communication latency because remote memory accesses do not take effect until after the execution of a synchronization. Second the parameter g enforces bandwidth limitations. It requires that messages be sent at most once every g arithmetic operations.

The BSP model is notable for those attributes which it does not incorporate. For example, the model does not charge overhead for a message to be injected into the network. Unlike communication overhead, the travel time of a message can be hidden by performing local computation which does not involve remote memory accesses. By not modeling overhead there is a strong incentive for latency hiding because all communication costs can be effectively hidden with enough parallel slackness in the program. The model also does not include any notion of processor topology, effectively removing the onus of performing data placement and describing explicit communication patterns and the potential for topology-based optimization.

Another example of a bridging model which has focused on more accurately reflecting existing machine attributes is the LogP model [12]. This model, though closely related to the BSP model, is distinct in two ways. First, it models asynchronous execution. The parameter L (now a capital for proper acronym formation) used to enforce bulk synchrony in the BSP model is used in the LogP model strictly as a measure of the message latency. Second, the model adds a new parameter, o, which captures the length of time required for a processor to inject a message into the network or receive a message from the network. This parameter, in essence, measures the dead time, lost processor cycles which cannot be captured with latency hiding techniques.

The development of the LogP model is notable in another sense: it is the product of efforts by a diverse group of researchers from theoretical, software and hardware disciplines suggesting the mutual benefits to be reaped with a unifying paradigm. The CTA model, the BSP model and the LogP model are representative of a recent trend to develop a consistant view of execution engines and programming models which can facilitate clear algorithm designs, efficient translations and high performance compilations.

Summary

The computational models presented above were chosen to be representative subset of the numerous proposed abstract models of parallel computation. The sheer volume of proposed models is sufficient testimony to both the lack of consensus and the perceived need for a unifying model, or set of models.

The survey suggests that evaluated individually, no single model seems to be acceptable. Evaluated as a group, however, they suggest that a small number of machine characteristics are the focus of the majority of the models. They include **Computational Parallelism**, **Communication Latency**, **Communication Overhead**, **Communication Bandwidth**, **Execution Synchronization**, **Memory Hierarchy**, and **Network Topology**.

These characteristics reflect the perspective of those whose primary objective is the design of efficient algorithms and clarity in the design process. Yet a unifying computational model must also address the requirements of those whose objectives are programming, implementation and machine design. A similar survey of the development and use of models in these communities was performed but due to space limitations cannot be presented here. The survey of these areas also suggested that within each area, on the whole, the models focused on a relatively small set of machine characteristics. The section below attempts to synthesize the results of the broader survey and present the implications for the development of a unifying computational paradigm.

3 A Case for a Simple, Prescriptive Model

The processes of designing solutions, encoding them in software, translating the machine independent software in to efficient executable instructions, and designing powerful execution hardware have many distinct modeling requirements. Yet these processes share common performance objectives. This commonality motivates the development of a unifying abstract machine model which can help coordinate these processes. To unify disparate disciplines the model must be simple, without sophisticated concepts from any single discipline yet incorporating the performance metrics of common interest.

Simplicity is also essential for an effective design process. To ensure clarity of focus a model which facilitates design can only address a few characteristics. While augmenting a simple model with ancillary machine characteristics and incentives may increase its descriptivity, it can clutter the design platform, obscuring primary characteristics.

To unify and coordinate, an abstract machine model needs to facilitate feedback to the machine design process itself. In addition to providing a clean design paradigm, the model should provide insight into what software or platform characteristics would enable efficient solutions for different classes of problems. It is with this philosophy of *simplicity* and *descriptivity* balanced with *prescriptivity* that the following sections discuss candidate model characteristics.

1. Computational Parallelism

To quantify parallelism most models include the number of physical processors, P, as a parameter. Though this seems straightforward, the issue of static versus dynamic parallelism complicates things. In problems with irregular or uncertain (non-oblivious) parallelism, designing a problem solution and encoding it becomes easier if one has the ability to create virtual parallel processes to be assigned to different tasks which may arise as the computation proceeds. This, however, leaves design of the means to exploit the parallelism entirely to the software and machine designers. Alternatively, allowing only a fixed number of physical processors leaves the task of scheduling parallel computation under uncertainty to the algorithm designer.

2. Latency

The inevitable physical separation of processing and memory elements in a scalable parallel computer creates a significant time delay when obtaining data from non-local memory. Though there is general agreement on the existence of this delay, there is less agreement on how to quantify it.

The simplest cost mechanism is to assume that all messages are of fixed constant length and incur a constant number, I units, of time delay. This simple choice provides an incentive for one-level reference locality but ignores other aspects of communication latency such as topology and the fixed and variable cost components of message transmission.

The current argument to ignore topology is to a great extent an artifact of high fixed **overhead** on the current generation of massively parallel computers. Overhead costs such as message formation and packet injection are so high that they swamp variable communication costs. Yet this overhead severly restricts the classes of programs that can run efficiently on these machines. If the goal is general purpose parallel computing, the overhead must be driven down to more reasonable ranges. One optimization technique to reduce the impact of high overhead is to facilitate the transmission of large, **variable-length messages**. By bundling messages the high fixed overhead can be amortized over a large number of bytes. But the inclusion of variable length messages in a model, though it captures this data parallelism, complicates the model. This optimization is likely to work for several classes of problems but should probably be considered an algorithm design optimization.

In response to the incentives to reduce communication overhead in parallel computers several machines have been envisioned in which more of the network communication mechanisms are handled in hardware, e.g. the M-machine [13]. Such decreases in the overhead of a communication increase the relative importance of network topology. Yet **contention**-based latency is a complicated function of **topology** and the communication pattern, and while toplogical considerations may provide performance optimizations, adding topology to a simple abstract machine model complicates it. Perhaps the ability for this kind of algorithm refinement could be incorporated in a more refined special purpose model.

The disparity between the speed of on-chip events and network events in the current generation of massively parallel computers motivates the simplification of an abstract machine model to include only a two-level **memory hierarchy**. Though this disparity may decrease, the utility of including a multi-level hierarchy is high only in problems which involve the movement of a significant amount of data. In these problems it can be advantageous to exploit data movement parallelism. But again, models including memory hierarchy erode the simplicity of a model unacceptably for general usage. Hierarchical models could be viewed as refinements to guide data movement optimization for special classes of algorithms.

The strategy of assuming that any communication incurs a loss of I units of processor time ignores the opportunity for **latency hiding**. Latency hiding is an optimization technique which could be important in the current generation of high latency massively parallel computers. Unfortunately this technique does not cloak the fixed overhead component of communication costs. Thus on computers with high communication overhead the importance of latency hiding diminishes. If there are substantial decreases in overhead, latency hiding could have more practical impact. Yet to realize substantial benefits from latency hiding requires software mechanisms such as non-blocking asynchronous communication mechanisms, very lightweight threads, or hardware support mechanisms which are still non-standard. (This is in addition to the availibility of lots of slackness.) In addition, including latency hiding mechanisms in a computational model runs the risk of distracting the algorithm designer from more substantive ideas for elegant and efficient design. Thus, the uncertain potential gain must be weighed against the risk of design obfuscation and the general burden of working within a more complicated model. It appears more reasonable to vest the incentives for latency hiding in the programming model. In this environment more informed choices can be made on the closely related issues of process *granularity* which depends on the characteristics of the software system such as the cost of a context switch, the cost of process creation, and the underlying mechanisms which support interprocess communication.

3. Bandwidth

With low communication overhead, latency hiding could become an important optimization. Yet incentives for latency hiding can also provide incentives to flood the network with messages. Bandwidth is a limited resource. In the current generaion of massively parallel computers, bandwidth limitations constrain the classes of problems which can be efficiently implemented on these platforms. Thus there are strong motivations to include bandwidth constraints in an abstract machine model.

There is a focus in hardware development on reducing the slope of the bandwidth hierarchy in parallel computers. Cost considerations as well as scalability issues make flat bandwidth hierarchies like those in older vector machines or the more recent Cray C-90 somewhat impractical. It is reasonable to assume that newer machines will seek to correct the bandwidth cliffs of the current generation and present a more tapered hierarchy.

It seems reasonable to conjecture that steep bandwidth hierarchies will be mitigated because machine designers have strong motivation to correct any binding constraint that severely limits the functionality of the computer. This balance conjecture essentially posits that machines will be designed so that no impediment to high performance computation or communication will be severely out of line relative to any other in the machine. In addition, if this is not actually the case one could argue that this should be the prescriptive input from algorithm designers if general purpose parallel computing is to be realized. Consistant with this conjecture is the idea that both latency and bandwidth constraints can be captured by the same model parameter. That is, one can charge I steps as latency for every remote access and prohibit more than one remote access every I steps.

As throughput approaches the network threshold, the relationship between message throughput and latency changes drastically. The use of one parameter would have to correspond to a bandwidth value which avoids extreme situations. With this caveat, the interval I would be a function of a given class of machines, set according to observed values. Therefore using the same paramenter, (I, the interval) to capture both constraints is a reasonable approach from the standpoint of descriptivity, as well as simplicity and prescriptivity.

4. Synchronization

Synchronous execution is an unreasonably rigid assumption in current or envisioned massively parallel computers. The ability to be effective platforms for many computationally intensive problems will require the ability to exploit irregular or asynchronous parallelism. The cost of synchronizing a subset of processes is related to the cost of communication. At a minimum, assuming no global synchronization mechanism, a synchronization entails at least one communication. Thus if only one parameter is incorporated in the model, I (the interval), one idea is use this same parameter to provide a cost measure for synchronization. Namely, allow at a maximum only one synchronization every I steps. Coupling these parameters also simplifies the model by avoiding the introduction of more concepts such as supersteps or phases.

While there is compelling motivation for the development of a unifying parallel machine model, the standards of simplicity and asymptotic analysis set by the RAM model make this difficult, expecially in an environment which is inherently more complicated and still evolving. Computational models no longer appear to be the exclusive domain of the algorithm designer. Both software and hardware designer have stronger motivations to focus on performance measures. This makes the task of developing a unifying model even more difficult (and more necessary): there are potential users from disparate disciplines with different objectives. Thus, to develop a model which is unifying the demands of these disciplines must be heeded.

To design an appropriate model requires that a set of characteristics be distilled which combine the needs and constraints of different domains and that they be incorporated in a simple but descriptive manner. Our tentative conclusion is that it may suffice to use a model with only two parameters: P, the number of processors and I, the communication interval. The interval is set to the binding constraint, the maximum of the latency, bandwidth or synchronization costs. In addition to simplicity, the justification for using only one parameter lies in the supposition or prescription that machine design will or should seek to balance the cost of these attributes, as the absense of balance can severely restrict the classes of applications that can perform well. Combining latency measures, bandwidth constraints and synchronization costs in one parameter, the model becomes simple and prescriptive, yet only somewhat less descriptive than other proposed models.

Acknowledgements

We would like to express our appreciation for the many insights provided by Larry Snyder, Kai Li, and Suresh Jagannathan, and to thank to Satish Rao for providing the inspiration.

References

[1] Aggarwal, A., Alpern, B., Chandra, A., Snir, M., "A Model for Hierarchical Memory", *Proc. of the 19th Annual ACM Symp. on Theory of Computing*, ACM, pp. 305-314, May (1987).

[2] Aggarwal, A., Chandra, A., and Snir, M., "Hierarchical Memory with Block Transfer", *Proc. of the 28th Annual IEEE Symp. on Foundations of Computer Science*, pp. 204-216, (1987).

[3] Aggarwal, A., Chandra, A., and Snir, M., "Communications Complexity of PRAMs", *Theoretical Computer Science* , Vol. 71, pp. 3-28, (1990).

[4] Aggarwal. A., Chandra, A., and Snir, M., "On Communications Latencies in PRAM Computations", *Proc. of the 1st Symp. on Parallel Algorithms and Architectures*, pp. 11-21, (1989).

[5] Akl, S., "Memory Access in Models of Parallel Computation: From Folklore to Synergy and Beyond", *Proc. of the 2nd Workshop on Algorithms and Data Structures*, Springer-Verlag, Berlin, pp. 92-104, (1991).

[6] Barnoy, A., and Kipnis, S., "Designing Algorithms in the Postal Model for Message Passing Systems", *Proc. of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 13-22, (1992).

[7] Blelloch, G., "Scans as Primitive Operations", *IEEE Transactions on Computers*, vol. 38, pp. 1526-1538, November (1989).

[8] Blelloch, G., Leiserson, C., Maggs, B., Plaxton, G., Smith, S., Zagha, M., "A Comparison of Sorting Algorithms for the Connection Machine CM-2", *Proc. of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 3-16, (1991).

[9] Cook, S., and Reckhow, R., "Time Boundend Random Access Machines", *Journal of Computer and Systems Sciences*, Vol. 7, pp. 354-375, (1973).

[10] Cole, R., and Zajicek, O., "The APRAM: Incorporating Asynchrony into the PRAM Model", *Proc. of the 1st Annual ACM Symposium on Parallel Algoritms and Architectures*, pp. 158-168, (1989).

[11] Cole, R., and Zajicek, O., "The Expected Value of Asynchrony", *Proc. of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 85-94, (1990).

[12] Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K., Santos, E., von Eicken, T., "LogP: Towards a Realistic Model of Parallel Computation", *Proc. of the ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming*, pp. 1-12, (1993).

[13] Dally, W., Keckler, S., Carter, N., Chang, A., Fillo, M., Lee, W., "M-Machine Architecture v1.0", MIT Concurrent VLSI Architecture Memo 58, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, (1994).

[14] de Torre, P., and Kruskal, C., "Towards a Single Model of Efficient Computation in Real Machines", *Proc. of Parallel Architectures and Languages Europe (PARLE91)*, Lecture Notes in Computer Science, Springer-Verlag, (1991).

[15] Gibbons, P., "A More Practical PRAM Model", *Proc. of the 1st Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 158-168, (1989).

[16] Gibbons, P., Matias, Y., and Ramachandran, V., "The QRQW PRAM: Accounting for Contention in Parallel Algorithms", *Proc. of the 6th Annual Symposium on Parallel Algorithms and Architectures*, pp. 638-648, (1994), to appear.

[17] Goodrich, M., "Parallel Algorithms Column 1: Models of Computation", SIGACT News, vol. 24, pp. 16-21, December (1993).

[18] Heywood, T., and Ranka, S., "A Practical Hierarchical Model of Parallel Computation: 1. The Model", Journal of Parallel and Distributed Computing, Vol. 16, pp. 212-232, (1992).

[19] Hightower, W., Prins, J., and Reif, J., "Implementations of Randomized Sorting Algorithms on Large Parallel Machines", *Proc. of the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 158-167, (1992).

[20] Jaja, J., "An Introduction to Parallel Algorithms", Addison-Wesley, (1992).

[21] Kruskal, C., Rudolph, L., and Snir, M., "A Complexity Theory of Efficient Parallel Algorithms", Theoretical Computer Science, Vol. 71, pp. 95-132, (1990).

[22] Leighton, T. "Introduction to Parallel Architectures: Arrays, Trees, Hypercubes", Morgan Kaufmann, San Mateo, CA (1992).

[23] Leiserson, C., and Maggs, B., "Communication-Efficient Parallel Algorithms for Distributed Random-Access Machines", *Algorithmica*, Vol. 3, pp. 53-77, (1988).

[24] Liu, P., Aiello, W., and Bhatt, S., "An Atomic Model for Message Passing", *Proc. of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 154-163, (1993).

[25] Mansfield, E., "Microeconomics", Third Edition, Norton and Co., New York, (1979).

[26] Martel, C., Subramonian, R., and Park, A., "Asynchronous PRAM Algorithms for List Ranking and Transitive Closure", *Proc. of the 31st IEEE Symposium on Foundations of Computer Science*, pp. 590-599, (1990).

[27] Melhorn, K., and Vishkin, U., "Randomized and Deterministic Simulations of PRAMs by Parallel Machines with Restricted Granularity of Parallel Memory", Acta Informatica, vol. 21, pp. 339-374, (1984).

[28] McColl, W., "General Purpose Parallel Computing", Technical Report, NEC Research Institute, Princeton, NJ, April (1992).

[29] Siegal H., Abraham, S., Bain, W., Batcher, K., Casavant, T., DeGroot, D., Dennis, J., Douglas, D., Feng, T., Goodman, J., Huang, A., Jordan, H., Jump, R., Patt, Y., Smith, A., Smith, J., Snyder, L., Stone, H., Tuck, R., and Wah, B., "Report of the Purdue Workshop on Grand Challenges in Computer Architecture for the Support of High Performance Computing", *Journal of Parallel and Distributed Computing*, Vol. 16, pp. 199-211, (1992).

[30] Snyder, L., "Type Architectures, Shared Memory and the Corollary of Modest Potential", *Annual Review of Computer Science*, Annual Review, Inc. pp. 289-318, (1986).

[31] Stromberg, K.,"An Introduction to Classical Real Analysis", Wadsworth, Belmont, California, (1981).

[32] Upfal, E., "Efficient Schemes for Parallel Communication", Journal of the Association for Computing Machinery, Vol. 31, pp. 507-517, (1984).

[33] Valiant, L., "A Bridging Model for Parallel Computation", *Communications of the ACM*, Vol. 33, pp. 103-111, (1990).

[34] Valiant, L., "General Purpose Parallel Architectures", van Leeuwen, J., ed. *Handbook of Computer Science*, MIT Press, (1990).

[35] Valiant, L., "Why BSP Computers", Technical Report TR-26-92, Aiken Computation Laboratory, Harvard University, Cambridge, MA (1992).

[36] Mansour, Y., Nisan, N., Vishkin, U., "Trade-offs Between Communication Throughput and Parallel Time", *Proc. of the 26th Symposium on Theory of Computing*, Montreal, Quebec, Canada, pp. 372-380, 1994.

[37] Vitter, J., "Efficient Memory Access in Large Scale Computation", *Proc. of the 1991 Symposium on Theoretical Aspects of Computer Science (STACS)*, Lecture Notes in Computer Science, Springer-Verlag, pp. 26-41, (1991).

[38] Vitter, J., and Shriver, E., "Algorithms for Parallel Memory I: Two Level Memories", and "Algorithms for Parallel Memory II: Hierarchical Multilevel Memories", Technical Reports, CS-1993-01 and CS-1993-02, Department of Computer Science, Duke University, January (1993). (to appear in a special issue of Algorithmica on the subject of large scale memories; a summarized version of this research was presented at the 22nd Annual Symposium on Theory of Computing, Baltimore, MD, May 1990).