Models and Resource Metrics for Parallel and Distributed Computation†

Zhiyong Li, Peter H. Mills, John H. Reif

Department of Computer Science,
Duke University,
Durham, N.C. 27708-0129

Abstract

This paper presents a framework of using *resource metrics* to characterize the various models of parallel computation. Our framework reflects the approach of recent models to abstract architectural details into several generic parameters, which we call resource metrics. We examine the different resource metrics chosen by different parallel models, categorizing the models into four classes: the basic synchronous models, and extensions of the basic models which more accurately reflect practical machines by incorporating notions of asynchrony, communication cost and memory hierarchy. We then present a new parallel computation model, the LogP-HMM model, as an illustration of design principles based on the framework of resource metrics. The LogP-HMM model extends an existing parameterized network model (LogP) with a sequential hierarchical memory model (HMM) characterizing each processor. The result accurately captures both network communication costs and the effects of multileveled memory such as local cache and I/O. We examine the potential utility of our model in the design of near optimal sorting and FFT algorithms.

1. Introduction

While the emergence of a large number of highly parallel computers have vastly increased the performance potential for solving large-scale scientific and engineering applications, the effective design and implementation of algorithms for them remains problematic. The diverse architectures and interconnection networks of the parallel machines – such as the Intel Paragon, Thinking Machines CM-5, KSR1, and Ncube with interconnection networks of mesh, fat-tree, ring and hypercube respectively – have a pervasive impact on algorithm performance. Each architecture has distinct properties on which may depend the performance of algorithms. The problem is thus how to design the algorithms or software to accommodate the specifics of the various machines. One approach is to build practical parallel computation models – mathematical abstractions of a computing machine which hide the architecture details from the software designers – and use these models to guide the high-level design of parallel algorithms as well as to provide estimation of performance. The challenge then becomes to design a general model of parallel computation in such a way that it balances being sufficiently detailed to reflect realistic aspects impacting performance while still remaining abstract enough to be machine-independent and amenable to analysis [Ski91, Goo93].

Historically, the Parallel Random Access Machine (PRAM) is the most widely used parallel model [FW78]. The PRAM model assumes that all processors work synchronously and that interprocessor communication is essentially free. The PRAM model could be thought as one extreme which makes a large number of assumptions in order to simplify algorithm design. However, for many current parallel machines, the PRAM is often inaccurate in predicting the actual running time and resource utilization of algorithms since it hides details which impact performance such as the time required for network communication as well as issues of asynchrony and memory hierarchy. This proves significant given the current trend toward larger-grained asynchronous MIMD machines whose processors each may have their own sophisticated memory hierarchies and which communicate over relatively slow networks. The impact is that the actual performance of algorithms developed under the PRAM model often do not match analytical complexity.

This problem has spurred the development of several extensions of the PRAM which attempt to make the model more practical while still preserving much of its simplicity. The variations extend the PRAM to incorporate realistic aspects such as *asynchrony* of processes (e.g., the Phase PRAM [Gib89] and APRAM

†This work was supported under ARPA/SISTO contracts N00014-91-J-1985, N00014-92-C-0182 under subcontract KI-92-01-0182, Rome Labs Contract F30602-94-C-0037, and NSF Grant NSF-IRI-91-00681.

‡Authors' e-mail address: {zli,phm,reif}@cs.duke.edu

Preprint of paper appearing in Parallel Algorithms and Applications, Vol. 8, pp. 35-59, 1996.

[CZ89]), *communication costs* such as network latency and bandwidth (e.g., the LPRAM [ACS89], Postal Model [BNK92], BSP [Val90], and LogP [CKP+93]), and *memory hierarchy*, reflecting the effects of multi-leveled memory such as differing access times for registers, local cache, main memory and disk I/O (e.g., the P-HMM [VS94], PMH [AC94], and P-UMH [NV91]).

The approach followed by these models is that of a *parameterized* (or generic) model, which abstracts the architectural details into several generic parameters which we call *resource metrics*. Typical resource metrics include the number of processors, communication latency, bandwidth, block transfer capability, network topology, memory hierarchy, memory organization and degree of asynchrony. Using such a parameterized model one can design broadly applicable parameterized algorithms that can be tailored to specific machines by instantiating the parameters, such as latency and bandwidth, to match machine characteristics. The more recent models typically try to use more parameters to more finely capture the resource characteristics of parallel machines. However, a careful balance must be struck between incorporating detail and being too finely parameterized (in too many dimensions) so as to render optimal algorithm design impossible. Therefore, identifying resource metrics and appropriately choosing them is critical in the design of models of parallel computation.

In this paper we identify resources and resource metrics which are important for the performance of parallel machines and use them as a framework to characterize the variety of parallel models. Within this framework we will categorize models into four classes: basic synchronous models, asynchronous models, models which incorporate notions of latency and bandwidth, and models which address hierarchical memory. The models discussed here are generally in an increasing order of the number of resource metrics considered.

Throughout the discussion, we use the problem of Fast Fourier Transform (FFT) computation to illustrate the principles of algorithm design and complexity analysis for many of the different models. The data movement of the FFT computation forms a butterfly graph, also called a FFT graph. The N-point FFT graph with $N = 2^m$, can be defined as follows: there are N input and output points denoted as $x_{0,0}, x_{0,1}, ..., x_{0,N-1}$ and $x_{m,0}, x_{m,1}, ..., x_{m,N-1}$ respectively. The computation is often called pebbling and denoted as $x_{j,q} = f(x_{j-1,q}, x_{j-1,r})$, where f is a constant cost function and the only difference of q and r is in the $(j-1)$th position when represented as binary numbers.

The examination of the resource metrics chosen by previous models reveals a void in models that accurately treat both network communication and multi-level memory. As a simple example of the process of developing improved performance models, we propose a new hybrid model of parallel computation, the LogP-HMM model, whose evolution naturally fills this void. The LogP-HMM extends an existing parameterized network model (the LogP, with resource metrics of latency, bandwidth, and overhead on handling messages) with memory hierarchy at each processor (there following the sequential HMM model). The LogP-HMM represents a pragmatic refinement of parallel computation models within the framework of resource metrics to accommodate more detailed performance measures. We examine the potential utility of LogP-HMM model in the design of near optimal sorting and FFT algorithms. It turns out that one of the near optimal FFT algorithms, the hybrid layout method, can run optimally in the P-HMM model.

The remainder of this paper is organized as follows. Section 2 identifies resources and resource metrics for parallel computation. Section 3 discusses basic synchronous models. Section 4 discusses extensions of the basic synchronous models, including asynchronous models, models incorporating communication cost, and hierarchical models. In Section 5 we summarize the resource metrics chosen by different models of parallel computation, define the new LogP-HMM model, and present near optimal sorting and FFT algorithms for this model. We conclude by reviewing the critical role the careful choice of resource metrics plays in the design of parallel models, as demonstrated through the LogP-HMM, and discuss ongoing research.

2. Resource metrics

We first present definitions of resources, resource metrics and models of parallel computation.

Definiton 1 *A resource refers to an architectural feature that significantly affects the performance of a parallel machine. A resource metric is a measure of the corresponding resource, which could be quantitative or qualitative. The value of a quantitative resource metric is normally a multiple of the unit processor execution time.*

Definiton 2 *A computational model is an abstraction of a computing machine which is characterized by the choice of several resources and the corresponding resource metrics. A computational model may thus be identified with a set of resource metrics. Moreover, algorithms will be designed and analyzed based on these resource metrics.*

For example, a sequential computer is suitably characterized by the resources of sequential computation time and space usage. It is commonly accepted that the sequential computational models, such as RAM and its hierarchical memory extensions HMM, BT_f and UMH [AACS87, ACS87, ACF90], reflect these resources quite well and therefore provide a common base for sequential computation. Resource metrics for a practical parallel machine are far more complicated than those of sequential machines. We identify the following list of significant resources and resource metrics for parallel computation:

Number of processors P. A theoretical model normally assumes that there is an unlimited number of processors available, while a more practical model assumes a bounded number of processors, such as hundreds or thousands.

Memory organization. Machines may be characterized as having physically shared memory or distributed memory. The former often has a local cache. The latter typically uses explicit message passing primitives.

Communication latency. The costs of accessing local memory and global memory in a shared memory machine are quite different; similarly, in a distributed memory machine, the costs of accessing local memory and communication with other processors are quite different. Latency is one measure of the cost of global memory access.

Degree of asynchrony. Processors may run synchronously, semi-synchronously (loosely synchronously) or asynchronously. Semi-asynchrony refers to a computation that is divided into a sequence of independent execution phases; within each phase, the program runs asynchronously, but all of the processors are synchronized at the end of each phase (i.e., barrier synchronization occurs).

Bandwidth. Communication bandwidth and memory bandwidth are both limited in practice. Currently, communication bandwidth lags far behind internal processor memory bandwidth. The bisection bandwidth, defined as the bandwidth across a line that separates the network into two parts, is normally used for the bandwidth resource metric.

Overhead of a processor for message handling. The communication overhead is the time that the processor engages in sending and receiving a message. In most cases, the value is dependent on the communication protocol implemented in a practical machine. For example, in the CM-5 it could be a linear function of the message size [CKP+93].

Block transfer capability. In most architectures, a significant cost (latency) is incurred to access the first of a contiguous block of words, but after that, successive words can be accessed in unit time.

Memory hierarchy. Many sophisticated machines have several layers of memory with differing access times, such as register, cache, main memory and secondary memory. A model capturing this memory hierarchy can organize the memory as a tree or a sequence of layers with increasing sizes, where each node or level is parameterized by memory size, block size, and intermodule bandwidth.

Memory contention. Memory can be accessed as a block or a set of banks. When the bank is unit size, the memory locations may be accessed simultaneously. This assumption is adopted by most of the models discussed in this paper. Protocols for resolving conflict in concurrent memory access include EREW, CREW, CRCW and QRQW.

Network topology. The processors may be interconnected using a mesh, cube, fat tree, ring, or other topology. The most common metric for this resource is the diameter of the network.

In the subsequent sections we present a detailed discussion of each model and its choice of resource metrics. Because each resource has a metric associated with it, we will henceforth not distinguish resources and resource metrics.

3. Basic synchronous models

PRAM. The PRAM model extends the sequential RAM model by replicating the processor part. A PRAM machine is a set of sequential processors sharing a global memory and each having its own private unbounded local memory. A PRAM computation is a sequence of read, write and computation steps. All processors execute in lock-step, that is, they are synchronized before they execute the next instruction. The costs of memory access, either to local or global memory, and computation steps are uniform. The cost of synchronization is free. Several variations of the PRAM model use different protocols to handle simultaneous access of several processors to the same location of global memory. Protocols include EREW (exclusive read - exclusive write), CREW (concurrent read - exclusive write), and CRCW (concurrent read - concurrent write). The latter protocol can be further divided

into several classes by the semantics of the concurrent write. The most recent variation is QRQW [GMR94], which assumes that simultaneous access to the same memory block will be inserted into a request queue and served in a FIFO manner.

VRAM. Another extension of the serial RAM model is the Vector Random Access Machine (VRAM) [Ble90]. The VRAM is a serial random access machine with the addition of a vector memory, a vector processor, and vector input and output ports. Typical vector instructions include elementwise operations, data movement operations, scans, and packs.

Two measures, step and element complexity, can be derived for a problem of size N in the VRAM model. Step complexity (s) is the total number of instructions executed and element complexity (e) is the vector length per primitive instruction call, summed over the number of the calls. These values give a measure of the parallel time and the total work respectively. The PRAM complexity of an algorithm designed in the VRAM model can be derived as $O(e/P + s)$.

The PRAM has proven to be useful by permitting algorithm designers to focus on the structure of computational tasks rather than the architecture details of a currently available machine. A large number of efficient algorithms have been developed by exploiting its simplistic assumptions, but in practice, some of the architectural issues that the PRAM ignores are important.

4. Extensions of the basic models

The PRAM model provides an abstraction that ignores concerns such as asynchrony, communication delay and memory hierarchy. In this section we discuss several extensions of the PRAM model which incorporate some of these measures. The extensions may be viewed as adding more resource metrics to the PRAM model in order to gain improved performance measures.

4.1. Asynchronous models

Among the first extensions to the PRAM were the Phase PRAM and APRAM models, which incorporate some notion of asynchronous execution.

Phase PRAM. The Phase PRAM [Gib89] extends the PRAM model with semi-asynchrony. A Phase PRAM machine consists of a shared global memory, a set of P sequential processors, and a private local memory for each processor. The computational task is separated into a set of phases of asynchronous execution, each ended by an explicit barrier synchronization. The cost of global read, global write and local operations are the same constant. The cost of a synchronization step is dependent on the number of processors. Space limitations preclude presentation of a Phase PRAM algorithm design for FFT computation; an example may be found in [Gib89]. It is worth noting that a variant of the Phase PRAM, the Phase LPRAM model, accounts as well for the cost of communication latency.

APRAM. The Asynchronous PRAM (APRAM) is a "fully" asynchronous model [CZ89]. The APRAM model consists of a global shared memory and a set of processes with their own local memories. The basic operations executed by the APRAM process are called events. An APRAM computation is denoted as the set of possible serializations of events executed by the processes. A virtual clock is associated with each serialization. This virtual clock assigns a time $t(e)$ to each event e. The clock "ticks" when each process has executed at least one event. Events may be read and write events, which operate on the shared and local memory, or local events. All events are charged unit cost.

The pair [round complexity, number of processes] is used to measure the complexity of an APRAM algorithm, where a round is defined as the sequence of events between two clock ticks in a computation. The round complexity for a computation is defined to be the maximum number of possible ticks for that computation. For an algorithm the round complexity is defined as the maximum round complexity over all of the possible computations. An example of APRAM algorithm design using these measures for the problem of summation may be found in [CZ89].

4.2. Models incorporating communication latency and bandwidth

The models discussed here consist of two subclasses: the synchronous latency models such as the LPRAM and BPRAM which add the notion of latency into the PRAM model, and models such as the BSP and LogP which not only incorporate asynchrony and latency but also address the issue of bandwidth limitation.

LPRAM. The Local-Memory PRAM (LPRAM) model [ACS89] consists of a shared global memory and a set of processors with unbounded local memory executing in lock-step. The access protocol to global memory is CREW. At every time step, each processor can perform either a communication step, in which it can write and then read a word from the global memory, or a computation computation step, which is an operation which accesses at most two words from its local

memory.

BPRAM. An extension of the LPRAM, the Block PRAM (BPRAM), is described in [ACS90]. The BPRAM takes into account the reduced cost for transferring a contiguous block of data. The BPRAM model is defined with two parameters L (latency or startup time) and b (block size). The cost of accessing local memory is unit time. However the cost of transmitting a block of size b of contiguous locations from global memory is $L + b$.

Postal model. The Postal model [BNK92] is a distributed memory model with the constraint that the point-to-point communication has latency λ. Several elegant optimal broadcast and summation algorithms have been designed based on this model, which were then extended for the LogP model [KSSS93]. Algorithms other than broadcast and summation have largely not been presented for this model.

Bulk-Synchronous Parallel model. The BSP is a distributed memory model [Val90]. Like the Phase PRAM, the BSP is also a semi-asynchronous model because it requires synchronization after each "superstep", within which the processes can run asynchronously. The BSP model is described in terms of three elements: processes/memory modules, a router which delivers the messages between pairs of components and a synchronizer which synchronizes all or a subset of the components. Within the framework of resource metrics, a router is just an abstraction of network bandwidth and latency. A computational task in the BSP model consists of a sequence of supersteps. In each superstep, every component is allocated a task which contains some combination of local computation steps and message transmissions. The local computations, including reads and writes to local memory, are charged unit cost. The message transmission is accomplished by the router, which can send and receive a certain number of messages in each superstep (or in BSP terminology, the router can realize an h-relation). The cost of realizing such an h-relation is assumed to be $gh + s$ time units, where g can be thought as the reciprocal of the communication bandwidth and s denotes the startup cost or the latency. If the length of a superstep is L, then L local operations and a $\lfloor \frac{L}{g} \rfloor$ - relation message pattern can be realized. The parameters of the machine are therefore L, g and P (the number of processors). A FFT algorithm for the BSP is described in [Val90].

LogP model. The LogP model is motivated by current technological trends in high performance computing towards networks of large-grained sophisticated processors. The LogP model uses the parameters L (an upper bound of latency for transmitting a single message), o (the computation overhead of handling a message), g (a lower bound of time interval between consecutive message transmissions at a processor) and P (the number of processors) [CKP+93]. In contrast to the BSP model, it removes the barrier synchronization requirement (h-relation in BSP) and allows the processors to run asynchronously. The network of a LogP machine has a *finite capacity* such that at any time at most $\lfloor \frac{L}{g} \rfloor$ messages can be in transit from or to any processor. It can support shared or distributed memory.

The LogP model encourages well-known general techniques of designing algorithms for distributed memory machines including exploiting locality, reducing communication complexity and overlapping communication and computation. The LogP model also promotes balanced communication patterns by introducing the limitation on network capacity so that no processor is overloaded with incoming messages. Moreover, it is often reasonable to ignore the parameter of o in a practical machine, such as in a machine with low bandwidth (high g value). Examples using this strategy can be found in the FFT algorithm discussed below and also in [KSSS93].

FFT. The data layout and communication scheduling are two key aspects to achieving an effective algorithm for the FFT problem under the LogP model. Three methods of data layout are discussed in [CKP+93]. The cyclic layout assigns the ith row of the butterfly to the ith processor. The block layout places the first $\frac{N}{P}$ rows on the first processor, the next $\frac{N}{P}$ rows on the second processor, and so on. Under either layout, each processor spends $\frac{N}{P} \log N$ time computing and sends and receives $\frac{N}{P}$ messages, which needs $(\frac{gN}{P} + L) \log P$ time. The third method is called hybrid layout, which switches from cyclic to blocked layout at any column between the $\log P$-th and the $\log \frac{N}{P}$-th (assuming $N > P^2$). With this layout, each processor sends $\frac{N}{P^2}$ messages to every other processor, requiring only $g(\frac{N}{P} - \frac{N}{P^2}) + L$ time. Therefore, this method leads to an algorithm with running time $O(\frac{N}{P} \log N + (\frac{N}{P} - \frac{N}{P^2})g + L)$. which is within a factor $(1 + \frac{g}{\log N})$ of optimal.

The naive communication schedule stalls on the first send. The technique of overlapping computation and communication can be used to eliminate this stall for the large problem instances. The method introduced in [Sah92] staggers the different starting rows for different processors: processor i starts with its $\frac{iN}{P^2}$-th row,

proceeds to the last row, and wraps around. This leads an optimal algorithm for large problem instances and reasonable g value.

4.3. Hierarchical models

The Parallel Memory Hierarchy model (PMH) [AC94] and Parallel Hierarchical Memory model (P-HMM) [VS94] discussed in this section address the concerns of memory hierarchy in a parallel setting. The P-HMM primarily originates from considering in a parallel network the existence of secondary or disk memory. PMH on the other hand uses "memory hierarchy" as a more general technique to model not only the hierarchy within a processor but also the communication characteristics of a parallel machine.

Parallel Memory Hierarchy model. The PMH is a so-called generic model which defines a class of specific models [AC94]. In the PMH model, a parallel computer is modeled as a tree and each node of the tree is called a *module*. All of the leaf modules are used to denote the processors and the internal modules hold the data. A child module is connected to its parent by a unique channel with a certain amount of bandwidth. Data in a module is partitioned into blocks which are the basic unit of data transfer between the child and parent. Thus communication between two processors proceeds up the tree by some path and then down the tree to the target processor, somewhat resembling the *fat* tree architecture.

The model is characterized by four parameters specified for each module m: *blocksize* s_m, *blockcount* N_m which denotes the number of blocks in each module, *childcount* c_m which denotes the number of children for each module, and *transfer time* t_m which denotes the number of cycles used to transfer a block between the current module and its parent. To model a particular computer, one chooses a tree structure and values for the parameters appropriate for the machine's communication capabilities and memory hierarchy. Even though this model is termed a parallel memory hierarchy, the internal modules need not necessarily correspond to actual memory modules of the real machine; when modeling the CM-5 for example, many of the modules are used to capture the interprocessor communication capabilities [AC94].

The PMH model derives from the developers' experience in tuning code for memory hierarchy. Many sequential algorithms have been developed for the original sequential UMH model. However, perhaps because of the complexity and generality of the model, not too many algorithms have been developed for the PMH parallel model.

Parallel Hierarchical Memory model. The P-HMM model is also called the parallel I/O model [VS90, VS94]. It originates from the consideration that data must often reside in secondary storage rather than main memory; in a parallel setting this may involve the parallel access of multiple disks. Therefore it is necessary to design parallel algorithms which consider the possible data movement between main and secondary memory [AV88], and more generally which consider multiple levels of memory including register and cache.

In the P-HMM model, each processor has a memory hierarchy organized into discrete levels, much like the memory organization in the HMM, and all of P separate memories are connected together at the base level of each hierarchy. A further assumption is that the P hierarchies can each function independently. Communication between hierarchies takes place at the base memory level (level 1) which consists of location 1 from each of the P hierarchies. The interconnection network for the P base memory level locations is normally assumed to be a hypercube (or cube-connected cycles) so that the P records in the base memory level can be sorted in $O(\log P)$ time. The model can be extended to allow block transfer; the resulting model is called the P-BT model [VS94]. Several other extensions which use the UMH memory model and PRAM interconnection respectively are discussed in [NV91].

Two factors are critical for developing effective P-HMM algorithms: data placement and movement between the levels of memory hierarchy, and data movement among the processors.

FFT. We present the following P-HMM algorithm from [VS94], performing the N-input FFT when $N \geq P^2$.

1. Compute $\sqrt{N}$ $\sqrt{N}$-inputs FFTs. Assume that ith FFT is on the ith contiguous group of $\frac{\sqrt{N}}{P}$ tracks (or memory levels).

2. Shift the records in the kth hierarchy to hierarchy $1 + (k+\mathit{offset}-1) \bmod P$, where $\mathit{offset} = (i-1) \bmod P$ for ith group.

3. Shuffle the records to form $\sqrt{N}$ new contiguous groups of $\frac{\sqrt{N}}{P}$ tracks. For every $1 \leq i \leq \sqrt{N}$, the ith new group consists of the ith record from each of the original $\sqrt{N}$ groups.

4. Do $\sqrt{N}$ $\sqrt{N}$-input FFTs for the new groups.

When $P \leq N \leq P^2$, we first do $\frac{N}{P}$ P-input FFTs, followed by a shuffle-merge, and then do $\frac{N}{P}$-input FFTs.

	Proc	Synchrony/ asynchrony	Memory Organization	Latency	Bandwidth	Block Transfer	Overhead	Memory Hierarchy
PRAM	P	Synchronous	Shared					
VRAM	1	Synchronous	Vector					
LPRAM	P	Synchronous	Shared	√				
BPRAM	P	Synchronous	Shared	√		√		
Postal	P	Asynchronous	Distributed	√				
PHASE PRAM	P	Semi-asynch	Shared					
PHASE LPRAM	P	Semi-asynch	Shared	√				
APRAM	P	Asynchronous	Shared					
BSP	P	Semi-asynch	Distributed	√	√			
LogP	P	Asynchronous	Both	√	√		√	
PMH	P	Asynchronous	Distributed	√	√	√	√	√
P-HMM	P	Asynchronous	Distributed	√				√
H-PRAM	P	Semi-asynch	Both	√				
LogP-HMM	P	Asynchronous	Distributed	√	√		√	√

Table 1: **Resource metrics chosen by different models of parallel computation.**

The time required by this algorithm, $T(N,P)$, is explained as follows. Step 1 and 4 take $\sqrt{N}T(\sqrt{N},P) + \frac{N}{P}\log\frac{N}{P}$ time. The first term in this formula is easy to understand. The second term arises because the different $\sqrt{N}$ $\sqrt{N}$-inputs FFTs sit in different memory levels (or tracks). We need to move them into the lowest memory levels in order to recursively compute the $\sqrt{N}$-input FFT. Similarly, the shifted elements need to be brought to the base memory and to be transmitted by the network which connects the base memory. Therefore, the shift time is $O(\frac{N}{P}(\log P + log\frac{N}{P}))$. The shuffling can be done in the order of input size times the data movement in the hierarchy, which is $O(\frac{N}{P}\log\frac{N}{P})$. Therefore, we have following recurrence

$$T(N,P) = 2\sqrt{N}T(\sqrt{N},P) + O(\tfrac{N}{P}\log N),$$

which gives the result $O(\frac{N}{P}\log N\log\frac{\log N}{\log P})$. This matches the FFT lower bound in [VS94] and so is optimal.

5. LogP-HMM model

The resource metrics chosen by different parallel models discussed so far are summarized in Table 1. It is apparent that each of the models addresses differing aspects of resource usage, which suggests that resource metrics are appropriate tools to categorize and examine different models of parallel computation. Using the framework of resource metrics, we can also build new models by adding or deleting several resource metrics from the existing models. However, this flexibility potentially causes difficulty. If we consider each resource metric as a dimension in the design space of a machine's architecture, then we may introduce too many dimensions with which to deal. Therefore, a careful analysis and choice of resource metrics is necessary.

The LogP-HMM model, defined below, serves as an illustration of how the framework of resource metrics can be used to guide the design of pragmatically refined models that fill a need for more detailed performance measures. From an examination of the resource metrics of existing models, it is apparent that a void exists in parallel models that accurately treat both network communication (as does the LogP) and multi-level memory (as does the P-HMM). The LogP model does not address the problem of several layers of memory. Yet it is important to model the several layers of memories which exist in many machines, since differing access times to local cache and disk may strongly effect performance. On the other hand, models such as the P-HMM or P-UMH address in detail the resource of memory hierarchy, but not so much the accurate characterization of the communication network. These models normally use simple assumptions about the network, such as a PRAM connection or an abstracted hypercube connection.

The LogP-HMM model fills the gap by combining both kinds of models together (or one might say by adding more resource metrics into either model). We take the approach of extending an existing parallel model with memory hierarchy. The resulting model consists of two parts: the network part and the memory part. The network part can be any of the parallel models such as BSP and LogP, while the memory part can be any of the sequential hierarchical memory models such as HMM and UMH. In this paper we will focus on the extension of the LogP model with HMM, which we call LogP-HMM.

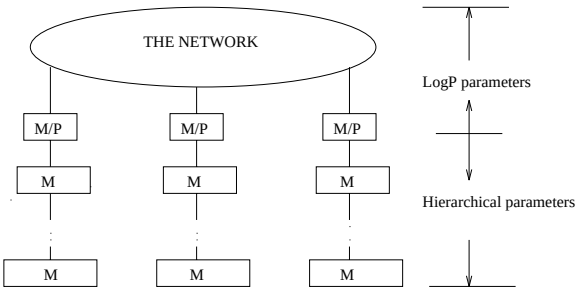

Figure 1: **Structure of the LogP-HMM model.**

5.1. Definition of the model

The LogP-HMM model, pictured in Figure 1, is defined much like the parallel hierarchy memory model [VS94]. A LogP-HMM machine consists of a set of asynchronously executing processors, each with an unlimited local memory. The local memory is organized as a sequence of layers with increasing size, where the size of layer i is 2^i. Each memory location can be accessed randomly; the cost of accessing a memory location at address x is $\log x$ (using access cost function $f(x) = \log x$). The processors are connected by a LogP network at level 0. In other words, the four LogP parameters, L, o, g and P, are used to describe the interconnection network. A further assumption is that the network has a *finite capacity* such that at any time at most $\lfloor \frac{L}{g} \rfloor$ messages can be in transit from or to any processor.

5.2. Algorithm design and analysis

Exploiting locality is the key to designing efficient algorithms for the LogP-HMM model. In LogP-HMM, there are two potential sources of data locality: the network part and the memory part. In the network part, we need to layout data in such a manner that each processor will use the data intensively before it needs the data in other processors. A similar situation exists in the memory part: before we move data to higher memory levels, the data should have been used intensively and should not be needed after moving. Several algorithms presented below illustrate these ideas. Before we present the algorithms, we first prove the following theorem.

Theorem 5.1 *The lower bound of sorting $N \geq P$ elements (or computing a FFT graph) in the LogP-HMM model with memory access cost $f(x)$=$\log x$ is*

$$\Omega(\frac{N}{P} \log \frac{N}{P} \log\log \frac{N}{P}).$$

Proof: In a LogP-HMM machine with P processors, the best way we can sort $N \geq P$ elements is to divide N elements into P sets of equal size and place the elements among the processors already in interprocessor-sorted order. Then each processor simultaneously sorts $\frac{N}{P}$ elements in its local memory and no communication between processors are required. In this case, the sorting lower bound for $\frac{N}{P}$ elements in each processor is also the lower bound for the whole sorting procedure. By the result in [AACS87], the sorting lower bound for $\frac{N}{P}$ elements in HMM model with memory access cost $f(x)$=$\log x$ is exactly $\Omega(\frac{N}{P} \log \frac{N}{P} \log\log \frac{N}{P})$. A similar argument can be used for FFT computation. Therefore we prove the theorem. □

FFT – Algorithm 1. Using the technique in [VS94] and the algorithm given in section 4.3.2, we can compute the FFT on a LogP-HMM machine. The time needed by this algorithm is

$$O(\tfrac{L+o}{L} g \tfrac{N}{P} \log \tfrac{N}{P} \log\log \tfrac{N}{P}).$$

Steps 1 and 4 take $\sqrt{N}T(\sqrt{N}, P) + \frac{N}{P} \log \frac{N}{P}$ time. The shuffling step spends time $O(\frac{N}{P} \log \frac{N}{P})$ after each group is "shifted" by an appropriate offset, because the shuffling can be done in the local memory hierarchy and does not cause any communication. The shifting can be done using following method. Each processor sends the "shift" messages to the other processors simultaneously. When $\lceil \frac{L}{g} \rceil$ messages have been sent, every processor begins to receive the $\lceil \frac{L}{g} \rceil$ messages. This procedure is continued until each processor has sent and received all of the $\frac{N}{P}$ messages. The time required for this communication is

$$O((\tfrac{N}{P} / \lceil \tfrac{L}{g} \rceil)(4L + 4o)) = O(\tfrac{L+o}{L} g \tfrac{N}{P}).$$

But each record sent may reside at a different level of the memory hierarchy. In order to move them into the lower memory levels, we need another $\log \frac{N}{P}$ factor, which gives

$$O(\tfrac{L+o}{L} g \tfrac{N}{P} \log \tfrac{N}{P}).$$

Therefore we have following recurrence

$$T(N, P) = 2\sqrt{N}T(\sqrt{N}, P) + O(\tfrac{L+o}{L} g \tfrac{N}{P} \log \tfrac{N}{P}),$$

which yields the bound given above. The algorithm is thus within a factor of $\frac{g(L+o)}{L}$ optimal.

FFT – Algorithm 2. Algorithm 1 basically uses block layout for the input data. This algorithm will use the hybrid data layout and a tighter upper bound can be derived. The idea of the hybrid method has been discussed in section 4.2. A more detailed discussion can be found in [Sah92].

Using the similar notation given in [Sah92], we denote $m = \frac{N}{P}$ and $l = \frac{m}{P}$. Two steps are used to compute the N-input FFT. In Step I, each processor computes a m-input butterfly and after each processor finishes its computation, it sends $\frac{N}{P^2}$ messages to each of the other processors. After each processor accepts the $\frac{N}{P} - \frac{N}{P^2}$ messages, it begins Step II which comprises the computation of the non-input nodes of l disjoint P-input butterflies. By the result of [AACS87], Step I computation needs $O(m \log m \log\log m)$ time. Step II computation needs $O(l * P \log P \log\log P) = O(m \log P \log\log P)$ time plus the time to move the l P-input FFTs to the lower memory levels, which is bounded by $O(P\frac{N}{P^2}\log\frac{N}{P})$ or $O(\frac{N}{P}\log\frac{N}{P})$. When there is no memory hierarchy, the communication time is

$$(\tfrac{N}{P} - \tfrac{N}{P^2})/\lceil\tfrac{L}{g}\rceil)(4L + 4o)) = \tfrac{L+o}{L} g(\tfrac{N}{P} - \tfrac{N}{P^2}).$$

However, the messages sent may reside at different memory levels. This gives the communication time to be

$$\tfrac{L+o}{L} g(\tfrac{N}{P} - \tfrac{N}{P^2}) \log \tfrac{N}{P}$$

Thus the total running time is:

$$O(\tfrac{N}{P}\log\tfrac{N}{P}\log\log\tfrac{N}{P} + \tfrac{L+o}{L} g(\tfrac{N}{P} - \tfrac{N}{P^2})\log\tfrac{N}{P}),$$

Because $\frac{N}{P} \geq P$, the running time is within a factor $1 + \frac{g(L+o)}{L \log\log\frac{N}{P}}$ of optimal.

Sorting. A near optimal sorting algorithm can be obtained by using columnsort [Lei85] and the modified median-sort algorithm proposed in [AACS87]. We will denote this modified median-sort algorithm as HMM-sorting in the rest of the paper and we will use the result that the HMM-sorting can sort N elements in $O(N \log N \log\log N)$ time. The columnsort performs sorting on a column-major $r \times c$ matrix with the conditions of $c\%r = 0$ and $r > 2(c-1)^2$. It executes eight consecutive steps, of which the odd-numbered steps sort the r elements in each column and the even-number steps permute the data among the processors.

In order to devise an efficient algorithm in the LogP-HMM model, we take $c = P - 2$ and $r = \frac{N}{P}$. The columnsort condition is satisfied if $N > 2P(P-3)^2$. Each processor will be responsible for sorting a column. Therefore, by using HMM-sorting, each of odd-numbered steps will take time $O(\frac{N}{P}\log\frac{N}{P}\log\log\frac{N}{P})$. In step 2 and 4, each processor sends and receives $\frac{N}{P} - \frac{N}{P^2}$ elements. These elements may reside at or send to different memory levels, therefore the communication overhead would be

$$\tfrac{L+o}{L} g(\tfrac{N}{P} - \tfrac{N}{P^2}) \log \tfrac{N}{P}.$$

In step 6 and 8, each processor sends and receives $\frac{N}{2P}$ elements. The communication overhead would be

$$\tfrac{L+o}{L} g \tfrac{N}{2P} \log \tfrac{N}{P}.$$

Therefore the sorting can be done in time

$$O(\tfrac{N}{P}\log\tfrac{N}{P}\log\log\tfrac{N}{P} + \tfrac{L+o}{L} g \tfrac{N}{P}\log\tfrac{N}{P}),$$

when $N > 2P(P-3)^2$, which is within a factor $1 + \frac{g(L+o)}{L\log\log\frac{N}{P}}$ of optimal.

5.3. An alternative optimal FFT algorithm for the P-HMM model

The hybrid algorithm discussed above can be adapted for the P-HMM model and an optimal running time can be achieved. The analysis is summarized below:

1. The time for m-input FFT is $O(m \log m \log\log m)$ on a single processor.
2. In the shuffle stage, each processor sends $\frac{N}{P} - \frac{N}{P^2}$ elements. In P-HMM, sending a message over the network takes $\log P$ time. Therefore, the time for this stage is $(\frac{N}{P} - \frac{N}{P^2}) \times (\log P + \log\frac{N}{P})$, where the second term of $\log\frac{N}{P}$ is for transferring the data in the memory hierarchy.
3. The time to compute l P-input FFTs is $(O(l(P \log P \log\log P) + \frac{N}{P}\log\frac{N}{P})$. Again the second term is for transferring the data in the memory hierarchy.

Summing all of them together, we get the following result:

$$O(\tfrac{N}{P}\log\tfrac{N}{P}\log\log\tfrac{N}{P}),$$

which matches the lower bound $O(\frac{N}{P}\log N \log\frac{\log N}{\log P})$ proven in [VS94]. And therefore it is optimal.

6. Conclusions

This paper presents a framework of using resource metrics to characterize the various models of parallel computation. Using the properties of resource metrics, we can classify models into the basic synchronous models and extensions which incorporate notions of asynchrony, communication costs, and memory hierarchy. The merits and disadvantages of these models are brought out by examining their characteristics in terms of resource metrics and their utility in designing such algorithms as FFT.

Resource metrics can be used not only to understand existing models, but also to guide the design of

new models. The LogP-HMM model, proposed in this paper, serves as an illustration of a model designed to fill the gap observed between models which address communication and those which address memory hierarchy. The design of near optimal sorting and FFT algorithms for LogP-HMM gives promise that the LogP-HMM model has the potential to serve as a viable tool for the design and analysis of what may prove to be practical algorithms for a large class of machines.

The LogP-HMM model leaves large room for further study. Ongoing investigations include examining the utility of replacing the memory part by a more practical model like the UMH. We are also pursuing the design of algorithms besides FFT and sorting for the LogP-HMM model. Finally, we are pursuing the comparative evaluation and experimental measurement of how accurately these models reflect real parallel machines by implementing the algorithms on machines such as the IBM SP-2.

References

[AACS87] A. Aggarwal, B. Alpern, A. Chandra, and M. Snir, "A model for hierarchical memory," in *Proc. 19th ACM Symp. on Theory of Computing*, pp. 305–314, May 1987.

[AC94] B. Alpern and L. Carter, "Towards a model for portable parallel performance: exposing the memory hierarchy," in *Portability and Performance for Parallel Processing*, pp. 21–41, John Wiley & Sons, 1994.

[ACF90] B. Alpern, L. Carter, and E. Feig, "Uniform memory hierarchies," in *Proc. 31st IEEE Symp. on Foundations of Computer Science*, 1990.

[ACS87] A. Aggarwal, A. Chandra, and M. Snir, "Hierarchical memory with block transfer," in *Proc. 28th Symp. on Foundations of Computer Science*, pp. 204–216, Oct. 1987.

[ACS89] A. Aggarwal, A. Chandra, and M. Snir, "On communication latency in PRAM computation," in *Proc. 1st ACM Symp. on Parallel Algorithms and Architectures*, pp. 11–21, 1989.

[ACS90] A. Aggarwal, A. K. Chandra, and M. Snir, "Communication complexity of PRAMs," *J. Theoretical Computer Science*, Mar. 1990.

[AV88] A. Aggarwal and J. Vitter, "The input/output complexity of sorting and related problems," *Comm. ACM*, vol. 31, pp. 1116–1127, Sept. 1988.

[Ble90] G. E. Blelloch, *Vector Models for Data-Parallel Computing*. MIT Press, 1990.

[BNK92] A. Bar-Noy and S. Kipnis, "Designing broadcasting algorithms in the Postal model for message-passing systems," in *Proc. 4th ACM Symp. on Parallel Algorithms and Architectures*, pp. 11–22, ACM, June 1992.

[CKP+93] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP: Towards a realistic model of parallel computation," in *Proc. 4th ACM Symp. on Principles and Practice of Parallel Programming*, pp. 1–12, ACM, 1993.

[CZ89] R. Cole and O. Zajicek, "The APRAM: Incorporating asynchrony into the PRAM model," in *Proc. 1st ACM Symp. on Parallel Algorithms and Architectures*, pp. 169–178, ACM, 1989.

[FW78] S. Fortune and J. Wyllie, "Parallelism in random access machines," in *Proc.10th ACM Symp. on Theory of Computing*, pp. 114–118, 1978.

[Gib89] P. B. Gibbons, "A more practical PRAM model," in *Proc. 1st ACM Symp. on Parallel Algorithms and Architectures*, pp. 158–168, ACM, 1989.

[GMR94] P. B. Gibbons, Y. Matias, and V. Ramachandran, "The QRQW PRAM : Accounting for contention in parallel algorithms," in *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pp. 638–647, 1994.

[Goo93] M. Goodrich, "Parallel algorithms column I: Models of computation," *SIGACT News*, vol. 24, pp. 16–21, Dec. 1993.

[KSSS93] R. Karp, A. Sahay, E. Santos, and K. Schauser, "Optimal broadcast and summation in the LogP model," in *Proc. 5th ACM Symp. on Parallel Algorithms and Architectures*, pp. 142–153, 1993.

[Lei85] T. Leighton, "Tight bounds on the complexity of parallel sorting," *IEEE Transactions on Computers*, vol. 34, no. 3, pp. 344–354, 1985.

[NV91] M. Nodine and J. Vitter, "Large-scale sorting in parallel memories," in *Proc. 3rd ACM Symp. on Parallel Algorithms and Architectures*, pp. 29–39, July 1991.

[Sah92] A. Sahay, "Hiding communication costs in bandwidth-limited parallel FFT computation," Technical Report UCB/CSD 92/722, UC Berkeley, 1992.

[Ski91] D. Skillicorn, "Models for practical parallel computation," *International Journal of Parallel Programming*, vol. 20, no. 2, pp. 133–158, 1991.

[Val90] L. Valiant, "A bridging model for parallel computation," *Comm. ACM*, vol. 33, pp. 103–111, Aug. 1990.

[VS90] J. Vitter and E. Shriver, "Optimal disk I/O with parallel block transfer," in *Proc. 22nd ACM Symp. on Theory of Computing*, pp. 159–169, 1990.

[VS94] J. S. Vitter and E. A. M. Shriver, "Algorithms for parallel memory II: Hierarchical multilevel memories," *Algorithmica*, 1994.