Performance Prediction for Data Intensive Applications on Large Scale Parallel Systems

Yuhong Wen, Geoffrey C. Fox
Northeast Parallel Architecture Center
Syracuse University
111 College Place
Syracuse, NY, 13244-4100, U.S.A
{wen,gcf}@npac.syr.edu

ABSTRACT

This paper presents a new interactive performance estimation tool – PetaSIM for large scale parallel systems. Our main approach is to divide the difficult performance estimation problem into three domains: application, software and hardware, to extract the system specifications and provide tools for the interactive changes of the system parameters over the Internet. Computers, networks and applications are described as objects with special attention to the proper representation of caches and hierarchical memories. PetaSIM represents a prototype of a performance specification language (PSL). We present encouraging initial results for a set of data-intensive applications form the real applications and discuss the extension of PetaSIM to support applications in distributed collaborative engineering.

Keywords: Performance Prediction, Performance Specification Language

1. INTRODUCTION

Large-scale data-intensive parallel applications have become the leading scientific computing problems on the massively parallel machines, because of their complexity and time-consuming. It will be great help to the application design and the new computer architecture design of petaflop performance if we can give the performance prediction before physically running the applications on the parallel machines. There are two kinds of performance prediction approaches, concept design level and detailed performance prediction. At the concept design, the goal is to provide a quick and roughly correct performance prediction at the early stage of model design of the application and/or the new computer architecture design. While the detailed performance prediction is aimed to provide the detailed information of a given application running on a specific computer system, normally, we call it performance simulation.

In this paper, we will present a performance estimator which address to the conceptual performance prediction, called PetaSIM. It is designed also with particular attention to easy interactive comparison of different system design. It is quite convenient to change application structure in PetaSIM but we have chosen to focus on cases where one has a relatively fixed application suit and wish to rapidly explore a range of system designs. A Java applet front-end is used to optimize interactive estimation.

The performance estimator PetaSIM is built around the performance prediction process sketched in Fig.1 [1]. The distinctive feather of our approach is the use of machine and problem abstractions which although less accurate than detailed complete representations, can be expected to be more robust and further quite appropriate for the rapid prototype needed in the design of new machines, software and algorithms. The hearts of this performance prediction process are two technologies - PSL (Performance Specification Language) and PetaSIM [1], [3]. In this paper, we will address the design of PetaSIM, which will

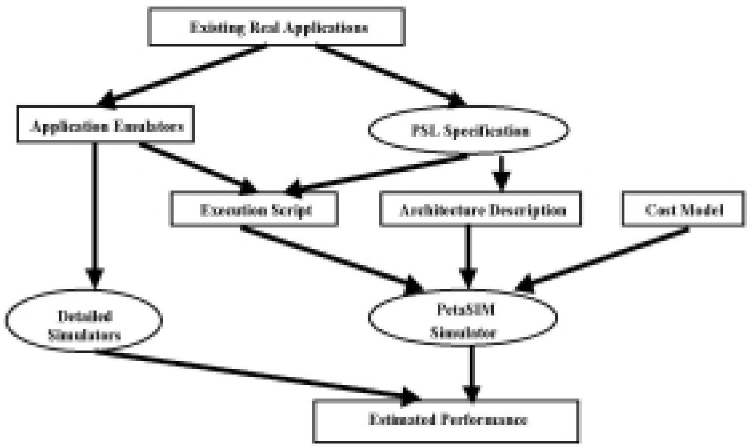


Fig.1: The Performance Prediction Process

take three key inputs from PSL, which describe respectively the target machines, application, script specifying execution of the application on the machine, to get an estimation of the performance of the application running on the machine. All kinds of research have showed that the performance prediction / estimation has been a very difficult problem, because of different kinds of applications, and different kinds of computer systems. It is important to design a general performance specification language (PSL) to support the performance estimation. In this paper, we will also show that PetaSIM is an initial step to suggest the characteristics of such a Performance Specification Language.

The rest of the paper is organized as the followings. Section 2 introduces the design and implementation of PetaSIM. In Section 3, we will give some real applications' performance estimation results running on IBM-SP2, using different architecture descriptions in PetaSIM. And Section 4 is the conclusion and further work.

2. PERFORMANCE ESTIMATOR -- PetaSIM

In this section, we will discuss the design and implementation of our performance estimator PetaSIM for parallel memory hierarchy system. PetaSIM takes three key inputs, which describe respectively the target machines, application, script specifying execution of the application on the machine, to get an estimation of the performance of the application running on the machine. PetaSIM takes both inputs from the application emulators (such as University of Maryland Emulators)[4] and hand written codes.

2.1 Emulators

An application emulator is a program to extract computational and data access patterns that closely resemble the patterns observed when executing a particular class of applications. In practice, an emulator is a simplified version of the real application, but contains all the necessary communication, computation and I/O characteristics of the application required for the performance prediction study. Using an emulator result in less accurate performance estimations than using full application, but it is more robust and enables fast performance predictions for rapid prototyping of new machines. An application emulator models the computation and data access patterns of the full application in a parameterized way. Adjusting the values of the parameters makes it possible to generate various application scenarios within a single class of applications.

In a simulation-based performance prediction framework, application emulator provides a specification of the behavior of the application to the simulator. Using an application emulator has several advantages over using traces from actual program runs or running the full application on the simulator. First, a trace is static and represents the behavior of the application for a single run on a particular configuration of the machine. Since an application emulator is a program that can be run on the simulator, it can model the dynamic nature of an application and can be used for different machine configurations. Second, running a real application may complicate the task of the simulator unnecessarily. By abstracting away parts of the application that are not critical to predicting performance, an application emulator can allow an efficient simulation without getting bogged down in the unimportant details of the application. Third, execution of a complete application requires the availability of real input data. Since the application behavior is only emulated, an application emulator does not necessarily require real input data, but can also emulate the characteristics of the actual data. This can enable performance studies of applications on large machine configurations with large data sets. An application emulator without a great amount of detail can be used for rapid prototyping of the performance of the application on a new machine configuration; while a highly detailed emulator can be used, for instance, to study different parallelization strategies for the application.

2.2 Petasim

There are three parts of descriptions in the *PetaSIM* performance estimation system, architecture description and application description, and the system/software description. The most general computer architectures can be specified using the PetaSIM *nodeset* and *linkset* objects while the applications can be specified using *dataset* and *distribution* objects. While the system description represent the software feature of the parallel machines.

A *nodeset* is a collection of entities with current types allowed as:

- *memory* with cache (with flushing rules) as special case
- *disk* which is essentially same as a memory.
- *CPU* where results can be calculated
- *pathway* such as a bus, switch or network cable which concentrates data

A *linkset* connects *nodesets* together in various ways. *distributions* specify the horizontal (geometrical) connectivity of *nodesets* and *linksets*. Typically these are arranged in a natural default for the classic homogeneous architectures. The default mapping is inferred from sizes of *nodesets* and done in a simple one-dimensional block fashion. The vertical (flow of information) connectivity in the architecture is specified in the execution script with defaults implied in architecture specification.

The application is specified by a *dataset* object, whose implementation is controlled by a *distribution* object that specifies classic HPF style geometric decomposition across memories and CPU objects. The computation is specified by the execution script, which also specifies data movement.

nodeset, *linkset*, *dataset* and *distribution* are Java classes that are subclassed as necessary to give particular special cases with particular capabilities. They have methods that are defaulted for simple cases but can be overridden for complicated cases. Thus we are essentially Java as IDL (Interface Definition Language) for these core PetaSIM objects.

2.2.1. *nodeset* Object Structure in PetaSIM Estimator

nodeset has the following properties:

- **name**: one per *nodeset* object
- **type**: choose from **memory, cache, disk, CPU, pathway**
- **number**: number of members of this *nodeset* in the computer architecture
- **grainsize**: size in bytes of each member of this *nodeset* (only relevant for memory cache or disk
- **bandwidth**: this is maximum bandwidth allowed in any one member of this nodeset
- **floatspeed**: performance on floating point arithmetic specified as a time to do a single operation for entities in cache. Only used by a CPU
- **calculate()**: method for CPU *nodesets* that performs computation implied by **floatspeed** and other architectural features.
- **cacherule**: controls persistence of data in a memory or cache
- **portcount**: number of ports on each member of *nodeset*
- **portname[]:** ports connect to *linksets* and a member of a *nodeset* has one or more ports -- each of which has a name. A port corresponds to a class of connections and depending on number of members involved, a given port can correspond to multiple connections
- **portlink[]**: name of *linkset* connecting to this port
- **nodeset_member_list**: list of *nodeset* members in this *nodeset* (for *nodeset* member identification)

2.2.2. *linkset* Object Structure in PetaSIM Estimator

A basic *linkset* has the following properties. A derived *linkset* object is gotten by concatenating several basic *linksets* objects together. Derived *linksets* could be specified by special scripts or just written directly in Java.

- **name**: one per *linkset* object
- **type**: choose from **updown, across**. If **across**, this is a network of given **topology**, linking members of a single *nodeset*. If **updown**, this is a link between two different *nodesets*
- **nodesetbegin**: name of initial *nodeset* joined by this *linkset*
- **nodesetend**: name of final *nodeset* joined by this *linkset*. **Nodesetend** and **nodesetbegin** are identical if type is **across**
- **topology**: used for **across** networks to specify linkage between members of a single *nodeset*
- **duplex**: choose from **full** or **half**. If **half** only allow transmission from **nodesetbegin** to **nodesetend**. If **full** allow either direction with bandwidth limiting sum of both directions.
- **number**: number of members of this *linkset* in the computer architecture
- **latency**: time to send zero length message across any member of this *linkset*
- **bandwidth**: this is maximum bandwidth in bytes per second allowed across any link in this *linkset*. Time T_{lk} to transfer information from *nodeset* ***l*** to *nodeset* ***k*** is expressed as ***latency + bandwidth*** number of bytes. Here ***l*** refers to **nodesetbegin** and ***k*** to **nodesetend**
- **send()**: method that calculates implications of sending information through given *linkset*. For a derived *linkset*, this method can include multiple references to properties and methods of basic *linksets* and *nodesets*.
- **distribution**: name of geometric distribution controlling this *linkset*
- **nodeset_member_list:** list of *nodeset* members in this *nodeset* (for *nodeset* member identification)

2.2.3. *distribution* Object Structure in PetaSIM Estimator

distribution has the following properties:

- **name**: one per *distribution* object
- **type**: choose from **block1dim, block2dim, block3dim** (can obviously add more to this in analogy with HPF) to specify geometrical structure of entity being distributed. Note most computer architectures are implicitly done as one-dimensional block *distribution*

2.2.4. *dataset* Object Structure in PetaSIM Estimator

dataset has the following properties:

- **name**: one per *dataset* object
- **type**: choose from **grid1dim, grid2dim, grid3dim**, specifies type of *dataset*
- **bytesperunit**: number of bytes in each unit. If 5 field values at each grid point and double precision used, then **bytesperunit** is 40
- **floatsperunit**: update cost as a floating point arithmetic count. Differences between double or single precision, should be reflected in values of *CPUnodeset.floatspeed* and *dataset.bytesperunit*
- **operationsperunit**: operations in each unit. A *dataset* contains *totalsize* of units, *operationsperunit* then reflects the operations in each unit for the *CPUnodeset* to calculate the computing time.
- **update():** ethod that updates given *dataset* which is contained in a CPU *nodeset* and with a grainsize controlled by last memory *nodeset* visited.
- **transmit()**: method that calculates cost of transmission of *dataset* between memory levels including either communication (between distributed nodes) or movement up and down hierarchy. Note classic grid problems are assumed to be implemented using ghost cells and that this involves the edges of regions being transmitted.

2.2.5. Computation/Communication Instructions (Execution Script)

Much of the execution is controlled by methods in *nodeset, linkset* and *dataset* objects. Some typical additional commands that implicitly invoke these methods are:

- **send** DATAFAMILY **from** MEM-LEVEL-L **to** MEM-LEVEL-K
- Here DATAFAMILY is a *dataset* specified by name
- MEM-LEVEL-K, MEM-LEVEL-L are *nodesets* labeled by name which must be linked by a *linkset.*
- **move** DATAFAMILY **from** MEM-LEVEL-L **to** MEM-LEVEL-K
- **Use** distribution DISTRIBUTION **on** NODESET1,…,LINKSET1,…,DATASET1
- **compute** DATAFAMILY-A,DATAFAMILY-B .. **on** MEM-LEVEL-L
- **synchronize** synchronizes all processors (loosely synchronous barrier). Pipelining which is normally assumed, is stopped by this.

3. EXPERIMENT RESULTS

In this section we summarize some preliminary results from PetaSIM with three data-intensive applications Pathfinder, Titan and Virtual Micro-Scope from the University of Maryland. The architecture and application description files are all automatically generated by University of Maryland's emulators[4].

3.1 Remote Sensing – Titan and Pathfinder

Titan is a parallel shared-nothing database server designed to handle satellite data. The input data for Titan are sensor readings from the entire surface of the earth taken from the AVHRR sensor on the NOAA-7 series of satellites. The satellite orbits the earth in a polar orbit, and the sensor sweeps the surface of the earth gathering readings in different bands of the electric-magnetic spectrum. Each sensor reading is associated with a position (longitude and latitude) and the time the reading was recorded for indexing purposes. In a typical operation for Titan, user issues a query to specify the data of the interest in space and time. Data intersecting the query are retrieved from disks and processed to generate the output. The output is a two-dimensional multi-band image generated by various types of aggregations operations on the sensor readings, with the resolution of its pixels selected by the query.

Titan operates on data-blocks, which are formed by groups of spatially close sensor readings. When a query is received, a list of data-block requests for each processor is generated. Each list contains read requests for the data-blocks that are stored on the local disks of the processor and that intersect the query window. The operation of Titan on a parallel machine employs a peer-to-peer architecture. Input data-blocks are distributed across the local disks of all processors and each processor is involved in retrieval and processing of data-blocks. The 2D output image is also partitioned among all processors, and each processor is responsible for processing data-blocks that fall into its local sub-region of the image. Processors perform retrieval, processing and exchange of data-blocks in a completely asynchronous manner. In this processing loop, a processor issues disk reads, sends and receives data-blocks to and from other processors, and performs the computation required to process the retrieved data-blocks. Non-blocking I/O and communication operations are used to hide latency and overlap these operations with computation. The data-blocks are the atomic units of I/O and communication. That is, even if a data-block partially intersects with the query window and/or the sub-region of the output image assigned to a processor, the entire data-block is retrieved from disk and is exchanged between processors.

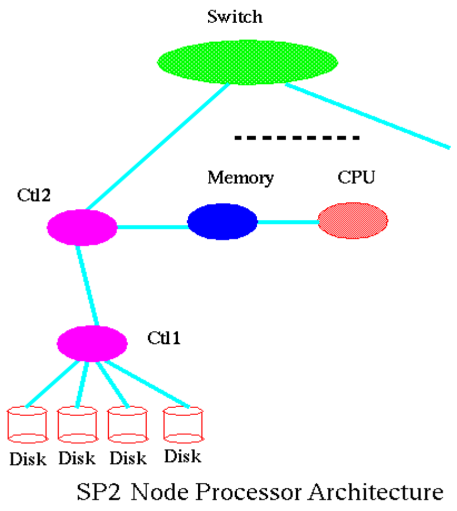


Fig.2: IBM-SP2 Simple nodeseet and linkset components -- Architecture I

Pathfinder is very similar to Titan except that it always processes all the input data that is available for a particular time period, over the entire surface of the earth. In addition, the operation of Pathfinder on a parallel machine employs a client / server architecture with separate I/O nodes and compute nodes.

3.2 Virtual Microscope

The Virtual Microscope is designed to emulate the behavior of a high-power light microscope. The input data for the Virtual Microscope are digitized images of full microscope slides under high power. Each slide consists of several focal planes. The output of a query into the Virtual Microscope is a multi-band 2D image of a region of a slide in a particular focal plane at the desired magnification level (less than or equal to the magnification of the input images). The server part of the software running on the parallel machine employs a peer-

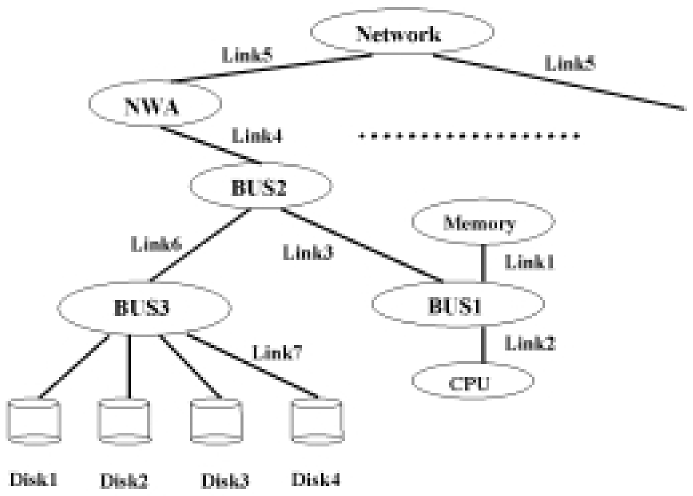
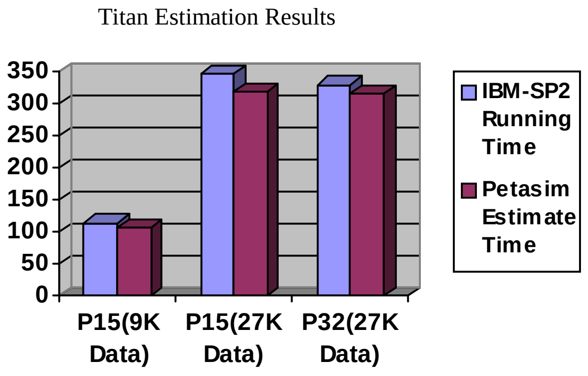


Fig.3: Detailed SP2 configuration used in fig. 5 showing the nodeset and linkset components – Architecture II

to-peer architecture. As in Titan and Pathfinder, input data is partitioned into data-blocks and distributed across the disks on the parallel machine. In a typical operation, multiple clients can simultaneously send queries to the server. When a query is received, each processor in the server retrieves the blocks that intersect with the query from its disks, processes these blocks, and sends them to the client. There is no communication or coordination between server processors. Different processors can even operate on different queries at the same time.

3.3 IBM-SP2 Architecture Description

We did the performance prediction experiments of the above real data intensive applications running on IBM-



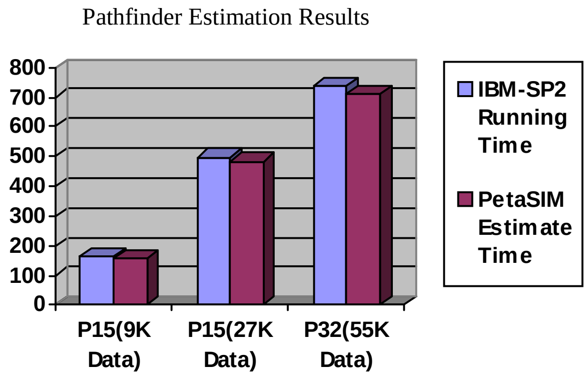


Fig. 4: PetaSIM Performance Estimation on Architecture I

SP2. In order to describe the features of the underlying parallel architecture, we abstract the memory hierarchy of each node on IBM-SP2 architecture in Fig.2 and Fig.3. We used two different kinds of architecture description in our experiments. Fig.2 gives a roughly description of the architecture, while in Fig.3, we did the performance estimation in a more detailed way.

Both pictures describe the architecture of one single node in IBM-SP2. It consists of *nodesets* joined by *linksets*. The whole system may contain 32, 64, or even more such nodes. PetaSIM also supports the description of heterogeneous architecture, the system may contain different kinds of nodes. Each node may not have the same architecture as the others.

3.4 Experiment Results

The results in Fig.4 show the application Titan and Pathfinder's performance comparison of PetaSIM estimate time with the real measured running time on IBM-SP2 based on the roughly description of the architecture I. The symbol P15(9K Data) represents that the application is running on 15 nodes of IBM-SP2, and the application has 9K data. The others are the same as this one. The results show that PetaSIM can get quite accurate performance estimation compared with the measured running time.

Fig.5 shows the PetaSIM estimation results on detailed description of architecture II. The results in Fig.5 are plotted against the number of parallel SP2 nodes except for one case (Pathfinder) where we also compare results plotted against number of I/O nodes in the system. From the benchmarks we can see that the PetaSIM estimate results are quite close to the measured application's running time.

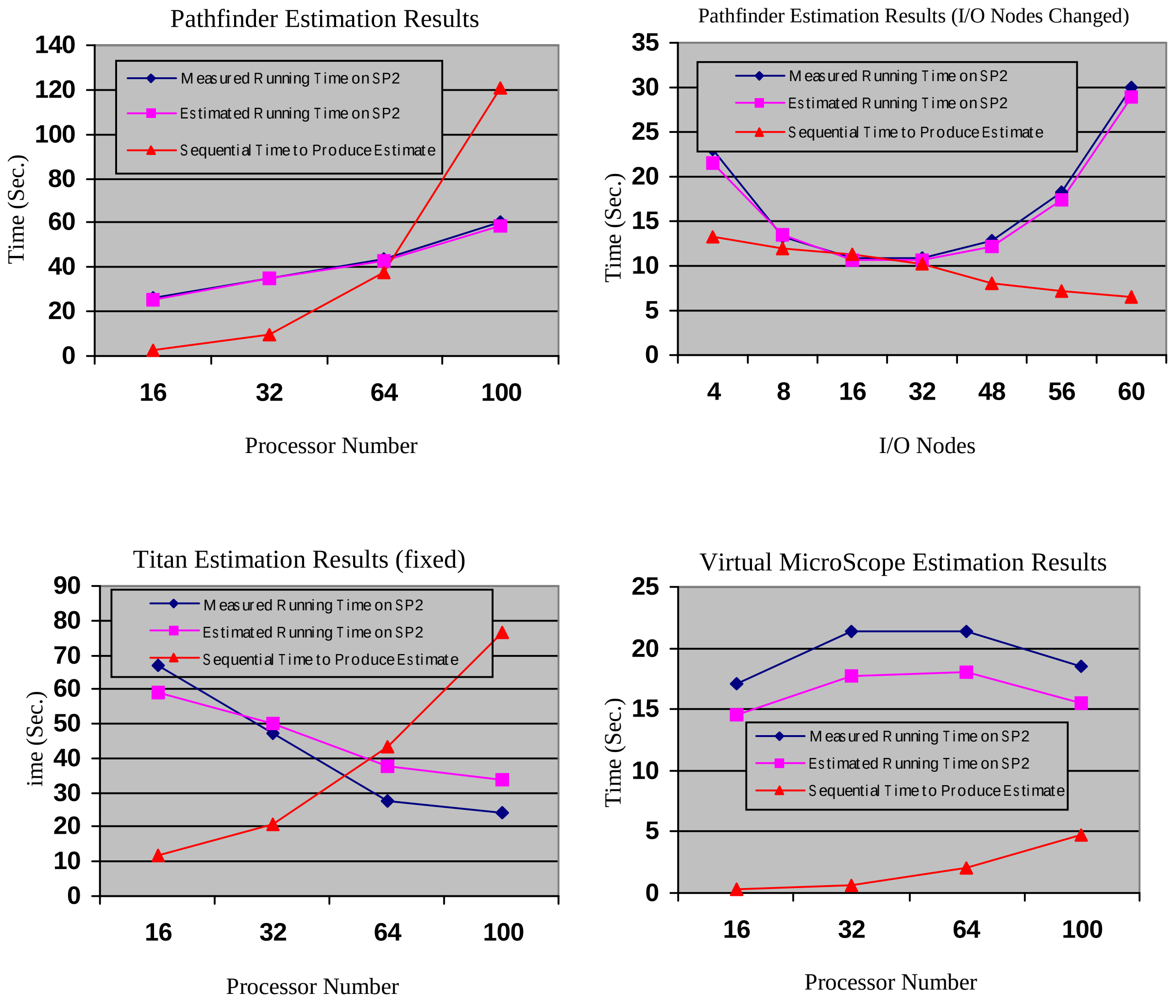


Fig.5: Measured Execution Time compared to the estimated execution time from PetaSIM for four examples. We also show the sequential execution time needed to produce the estimate. – Architecture II

These figures also show the actual wall clock time used by PetaSIM to produce the estimates. As this runs on a sequential machine and the execution script is not explicitly data-parallel, this time to get estimation increases linearly with the number of nodes. If one looks at the estimation time for simple data-parallel systems such as finite difference problems, PetaSIM would be much faster (as just a few not a few thousand lines of execution script) and take a time roughly independent of the number of nodes on the target machine. In fact we have recently abstracted the operations in the applications shown in Fig.5 to a “primitive data parallel operation” and have correspondingly drastically reduced the time needed to produce the estimate. One can expect to initially use a simple crude loop over parallel nodes for each new type of computation. If used enough, it can be implemented in data parallel fashion and added to PetaSIM’s library of operations.

PetaSIM provides the ability to easily modify the features of the architecture and application behavior, which helps greatly in the architecture conceptual design and get accurate performance estimation. PetaSIM also provides the interface for both inputs from the emulators (like our experience with the University of Maryland's emulators[4]) and from the hand-written code of the system designers, which make it even more flexible.

4. CONCLUSION AND FURTHER WORK

PetaSIM bases its performance estimate on several inputs: namely the computer architecture description, *nodeset* and *linkset*, and application description, *dataset* and *distribution*, and the data operation description, *execution script*, of the application. This has similarities to the approach used by the POEMS group led by University of Texas at Austin. In the POEMS system, one divides the performance estimation into application domain, system and software domain, and hardware domain. In each domain, they provide a model to describe the features of both application and architecture. The performance estimation will be based on the information provided. [6]

Compared with some other performance estimators, PetaSIM has some special characteristics. Most of the other simulators, such as the University of Maryland’s systems [4], [5], base their simulation on the task graph describing the application. PetaSIM instead uses an execution script for the application specified in ASCII format, which corresponds to a coarse grained description of the application. PetaSIM’s approach appears to provide a more intuitive interface to both application and resource description, which naturally supports rapid prototyping studies over a wide range of computer architectures. And because of the web-based technology used in PetaSIM, it has the advantage of easy to operate and change the system parameters, rapid and accurate performance estimation, and global availability.

In order to make PetaSIM more powerful to deal with more different kinds of real applications, we need to improve its abilities in representing both the applications, the computer architectures. In the application aspect, we are considering to provide more statements in the execution script to make it able to deal with more cases of data processing, such as loop operation. While in the computer architecture, we also need to provide more features to reflect the information of the parallel computer systems.

5. REFERENCES

[1] "The Petaflops Systems Workshops", Proceedings of the 1996 Petaflops Architecture Workshop (PAWS), April 21-25, 1996 and Proceedings of the 1996 Petaflops System Software Summer Study (PetaSoft), June 17-21, 1996, edited by Michael J. MacDonald (Performance Issues are described in Chapter 7).

[2] Kivanc Dincer and Geoffrey C. Fox, "Using Java in the Virtual Programming Laboratory: A web-Based Parallel Programming Environment", to be published in special issue of Concurrency: Practice and Experience on Java for Science and Engineering Computation.

[3] Geoffrey C. Fox and Wojtek Furmanski, Computing on the Web -- New Approaches to Parallel Processing-- Petaop and Exaop Performance in the Year 2007", submitted to IEEE Internet Computing, http://www.npac.syr.edu/users/gcf/petastuff/petaweb/

[4] Mustafa Uysal, Tahsin Kurc, Alan Sussman, Joel Saltz, Performance Prediction Framework for Data Intensive Applications on Large Scale Parallel Machines, University of Maryland Technical Report: CS-TR-3918 and UMIACS-TR-98-39, July 1998

[5] M. Uysal, A. Acharya, R. bennett, J. Saltz, "A Customizable Simulator for Workstation Networks", Proceedings of the International Parallel Processing Symposium, April 1997.

[6] Deelman, Bagrodia, Dube, Browne, Hoisie, Luo, Lubeck, Wasserman, Oliver, Teller, Sundram-Stukel, Vernon, Adve, Houstis, and Rice, POEMS: End-to-end Performance Design of Large Parallel Adaptive Computational Systems: Technical Report, August 1998