PetaSIM: A Performance Estimator for Parallel Hierarchical Memory Systems

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

Abstract

In both the design of parallel computer systems and the development of applications, it is very important to have good performance prediction tools. This paper describes a new approach -- PetaSIM, which is designed for the rapid prototyping stage of machine or application design. PetaSIM aims at good but not fully accurate results with a convenient applet interface for specification and interactive change of system parameters. Computers, networks and applications are described as objects in a Java IDL (Interface Definition Language) with special attention to the proper representation of caches and hierarchical memories. PetaSIM represents a prototype for a performance specification language or PSL. We present encouraging initial results for a set of data-intensive applications from the University of Maryland. We discuss the extension of PetaSIM to support applications of the type found in distributed collaborative engineering.

Keywords: Performance Estimation, Architecture Description, Performance Specification Language, Collaborative Engineering

1. Introduction

Performance prediction for grand challenge applications, such as those in weather forecasting, oil exploration or fundamental physics is an important and complex problem, especially when involving massively parallel computers where each node has a complex memory hierarchy. Again it is not easy to design or compare new computer architectures for petaflop performance where one must contrast divergent approaches and rapidly explore new concepts. Even today, designing computational grids to support distributed collaborative engineering is not easy because even if the system is conceptually simple, it is inherently dynamic and heterogeneous and one must support a system engineer making good strategic decisions on the network design and location of critical database and computational components. In each case, one does not typically requires precise performance estimates but rather results that are accurate to some 10-20% overall but allow more precision in comparing parameter choices. PetaSIM has been designed to address such problems with particular attention to easy interactive comparison of different system designs. 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 suite and wish to rapidly explore a range of system designs. A Java applet front-end [3,4] is used to optimize interactive estimation.

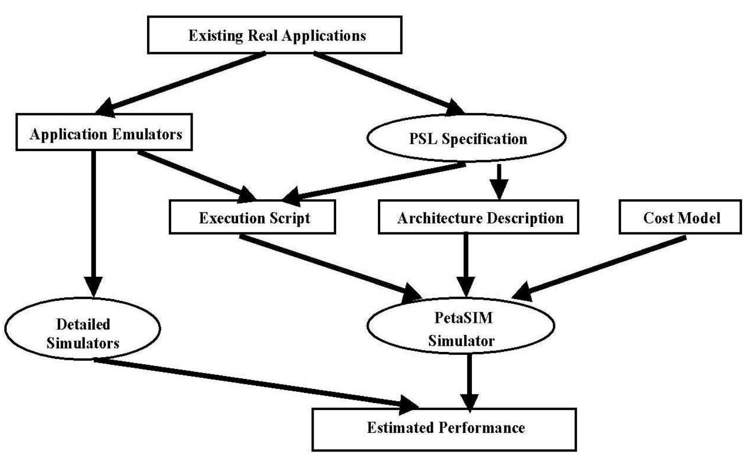

Fig.1: The Performance Prediction Process

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]. In this paper, we will address the design of PetaSIM, which will 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. Much research has shown that the performance prediction or estimation is a very difficult problem, because of the different kinds of applications, and multitude of distinct computer systems. [5-9] 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 (PSL). [10]

Section 2 describes the design features of PetaSIM while Sec. 3 gives a detailed description of the PSL (machine and application object structure) used to govern its execution. Sec.4 gives initial results from applications produced at the University of Maryland while Sec. 5 describes possible extensions of PetaSIM to support computational grids for collaborative engineering. Finally Sec. 6 describes future extensions to the basic infrastructure.

2. Motivation and Characteristics of PetaSIM

The performance estimator PetaSIM is aimed at supporting the (conceptual and detailed) design phases of parallel algorithms, systems software and hardware architecture. Originally PetaSIM was designed as a result of experiences at two workshops - PAWS and PetaSoft - aimed at understanding hardware and software architectures ten years from now when Petaflop (10^{15}) scale computing can be expected [1]. The most sophisticated PetaKernel [1] was a regular finite difference problem solved by a simple Jacobi iteration, which is of course well understood. However current tools did not even make estimates of the performance of these simple kernels easy. These workshops emphasized the need for tools to allow one to describe complex memory hierarchies (present in all future and most current machine designs) and the mapping of problems onto them in a way that allows reliable (if high level) performance estimates in the initial design and rapid prototyping stages.

PetaSIM is aimed at a middle ground - half way between detailed instruction level machine simulation and simple "back of the envelope" performance estimates. It takes care of the complexity - memory hierarchy, latencies, adaptability and multiple program components, which make even high-level performance estimates hard. It uses a crucial simplification - dealing with data in the natural blocks (or aggregates) suggested by memory systems – which both speeds up the performance simulation and in many cases will lead to greater insight as to the essential issues governing performance. We motivate and illustrate the design of PetaSIM by the well-known formulae for parallel performance of simple regular applications on nodes without significant memory hierarchy.[2] Then one finds:

$$\textit{Speed Up} = \textit{Number of Nodes}/(1 + \textit{Overhead})$$

Where the *Overhead* is proportional to $(\textit{Grain\_Size})^{-g}\,(t_{comm}/t_{calc})$ and in this case the natural data block size is the grain size or number of data points in each node. The power g measures edge over area effects and is $1/d$ for a system of geometric dimension d. (t_{comm}/t_{calc}) represents a ratio of communication to compute performance of the hardware.

Such a formula shows the importance of identifying the natural data blocks and how such high level analysis allows one to understand the relation of performance to memory sizing, I/O and compute capabilities of the hardware. PetaSIM generalizes this "back of the envelope" estimate to more complex problems and machines. It also includes not just primitive messaging performance (node to node as used in above estimate) but also collective (such as multicast) mechanisms, which are present in most applications but ignored in many simple analyses. Note that the simple performance estimate above is valid (with straightforward generalizations) on machines with the simple two level distributed memory hierarchy - namely memory is either on or off processor -which is essentially model built into the current generation of parallel programming systems as typified by HPF or MPI. As we described above, it is essential to generalize this machine model whether we want to provide input to either parallel programming tools or to performance estimation systems. Thus we believe our experience with PetaSIM will be very valuable in helping designing the new generation of parallel programming environments needed for the increasing complex high performance systems coming online.

We note one important design characteristic, which could be considered either a feature or a bug! Thus we deliberately ask the user to describe the data motion through the memory hierarchy even though this is likely automated in the actual hardware. This was in fact the preference expressed by the attendees at the motivating workshops [1] as when one is designing applications and hardware, it is useful to be able to know exactly what the data movement is. In future versions of PetasSIM this feature could be augmented by an option to automate this using particular cache management strategies.

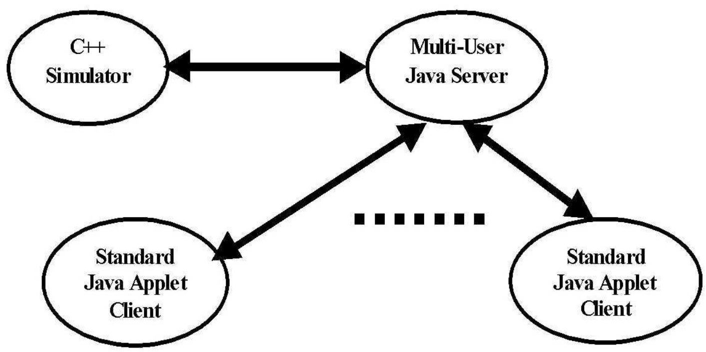

Fig.2: Architecture of PetaSIM

As well as a machine description, PetaSIM requires a problem description where we will use the PSL described in the following section. This essentially corresponds to a specification of the object structure of both problems and machines where the latter include compute, data and network components.

3. The Design of PetaSIM

3.1 Overall Architecture of PetaSIM

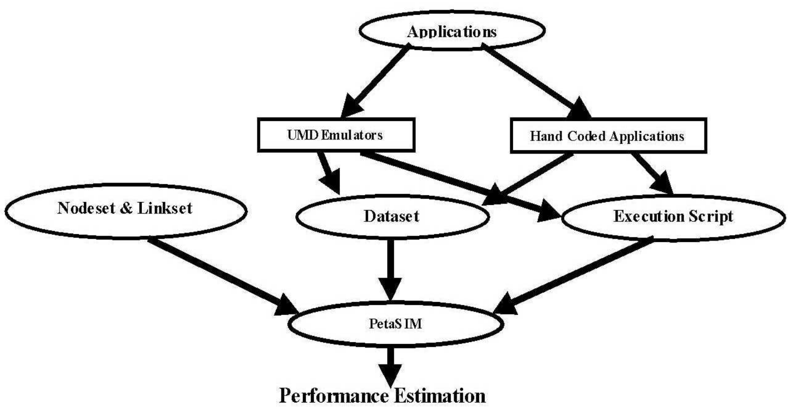

Fig.3 PetaSIM and Emulators and Hand Coded Applications

One of the most important motivations of PetaSIM is to provide a performance estimation tool for the new computer architecture designer to get a fast and accurate performance prediction during the conceptual architecture design. This requires that PetaSIM should be easy to operate and that it be convenient to modify features of the computer architecture. As shown in fig. 2, we chose a Java applet front end to meet this requirement in the design of PetaSIM. [3,4]

The heart of the PetaSIM is a C++ simulator which takes the computer architecture description and application description to give the performance estimation. A multi-user Java Server provides the service to different Java Applet clients from the global Internet. The designers can download the applet from web site (http://kopernik.npac.syr.edu:4096/PetaSIM/V1.0/PetaSIM.html) to get easy access to the PetaSIM. And as shown in fig. 3, PetaSIM supports both inputs from the emulators (such as University of Maryland Emulators) and hand written codes.

3.2 Specifications of PetaSIM

There are two parts of descriptions in the *PetaSIM* performance estimation system, architecture description and application 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.

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.

3.2.1. *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)

3.2.2. *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)

3.2.3. *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*

3.2.4. *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.

3.2.5. 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.

The difference between the ***send*** and ***move*** command is that ***send*** command will send the data from source node to the destination node, and then after the computing, send the data back to the original source. On the other hand, the ***move*** command will just move the data from source to destination.

3.3 Current Status of PetaSIM

A prototype of PetaSIM performance estimation system has been implemented in NPAC. It's based on the architecture description (*nodeset* class and *linkset* class) and application description (*dataset* class and *distribution* class, plus *execution script* to describe the operations on the *dataset*) as we discussed above. These two kinds of descriptions correspond to the Target Machine Specification, High Level Representation of Application and Execution Script in the design of PSL (Performance Specific Language), which defines the input for the *PetaSIM* performance estimator. PetaSIM has been used for some simple hand coded applications but the most interesting examples have come from interfacing it to the application emulators generated by the University of Maryland [11], which take task graphs corresponding to each application and generate simpler abstraction of the applications. After the initial tests with output from the University of Maryland's emulators, we found it necessary to provide the *PetaSIM* with the following features:

3.3.1. *nodeset* use in Application and Architecture Phases

In the current version, PetaSIM has the ability to specify the architecture's using *nodeset* and *linkset*, and so record the target machine's memory hierarchy. However we need the same information in the application emulator (or equivalent way of specifying execution). Thus we plicate the machine description in both the emulator generation and the PetaSIM architecture configuration files.

3.3.2. *dataset* hierarchy

In this version of PetaSIM, an application may use one or more datasets, each containing *totalsize* units of information. As a basic assumption of PetaSIM, all computation is supposed to be in a coarse grain block-block form where we take account of "real" blocks of nonzero size and the "virtual zero size blocks' formed by for instance ghost cells around the edge of domain. These blocks are formed from the *dataset* objects with a particular distribution on a particular memory *nodeset*.

Each statement in the execution script will operate on a specific *dataset*, for examples: **move** one *dataset* from one *nodeset* to another *nodeset*, or **compute** one *dataset* on a CPU *nodeset*.

In some applications, such as those considered in sec. 4, the applications may have a hierarchical structure. An application contains a set of *dataset*, and a *dataset* may contain a set of *sub-dataset*'s. In this case, a statement in execution script may operate on a specific *sub-dataset* or directly operate on a parent *dataset*. For example:

Suppose an application has *dataset1, dataset2*, and *dataset3*, and *dataset1* contains *sub_data1, sub_data2, sub_data3* and *sub_data4*.

Then the execution script may look like:

move *sub_data1* from *memory_node* to *cpu_node*
compute *sub_data1* on *cpu_node* ; which works on *sub_data1*
move dataset1 from *disk_node* to *cache_node*
move dataset1 from *cache_node* to *memory_node*
; which automatically operates on *sub_data1, sub_data2, sub_data3* and *sub_data4*

We provide a *dataset* hierarchy in the current version of *PetaSIM* and believe that is a generally useful feature of any PSL.

3.3.3. execution script library

Currently in the *execution script* of *PetaSIM*, we support four different kinds of statement, *send, move, Use distribution, compute* and *synchronize* commands, which are enough to express the essential features of data movement and computation operations in the applications. However this simple abstraction can make the execution script of applications very long; especially for realistic large applications such as those studied in sec. 4. So we intend to build up an execution script library, which contains aggregates of data movement and computation in a single function call. For example:

The calculation and data operations of *dataset* on *CPU_node*set could involve the following execution script statements:

move *dataset* from *disk_nodeset* to *disk_controller*
move *dataset* from *disk_controller* to *cache_nodeset*
move *dataset* from *cache_nodeset* to *memory_nodeset*
compute *dataset* on *CPU_nodeset*

In this case, we will make the execution script of applications shorter and easier to read and understand if we replace this by a single function call.

3.3.4. data dependency

In the current version of PetaSIM, the performance estimate for an application running on a specific machine will based on the features of target machines, *dataset* of the applications, and the execution script of the application. PetaSIM achieves its estimate by executing the script statements consecutively. Currently *PetaSIM* will not try to reorder the statements in the execution script. However in some cases, this is clearly inadequate as seen in the example:

s1: ***move*** *data1* from *node1* to *node2*
s2: ***move*** *data2* from *node3* to *node2*
s3: ***move*** *data3* from *disk2* to *node2* ; after the arrival of *data1*
s4: ***move*** *data4* from *disk2* to *node2* ; after the arrival of *data2*

In this example, s3 followed by s4 may not be the actual execution order as this depends on which of s1 and s2 finishes first.. If s1 finishes first, s3 may execute first on node2, and if s2 finish first, s4 may execute before s3. There is a data dependency between s3 and s1, s4 and s2. Currently PetaSIM doesn't have any mechanisms to deal with this kind of data dependency but we will try to support this later.

3.4 Interface of PetaSIM with PSL

Right now, PetaSIM takes architecture description and application description input from files. Here, we have 5 files, corresponding to *nodeset, linkset, dataset*, *distribution* description, and execution script, respectively. PetaSIM accesses these files using a standard socket connection.

In the architecture description of Machines, we have two parts, *nodeset* and *linkset*, which describe the detailed memory hierarchy and the linkage relationship between each hierarchy level. The *nodeset* description file contains all the information about each *nodeset*, and all of its *nodeset members*, and the linkage neighbors of each *nodeset member*. The *linkset* file contains the features of each *linkset*, and their *linkset* members, which link together two of the *nodeset* members.

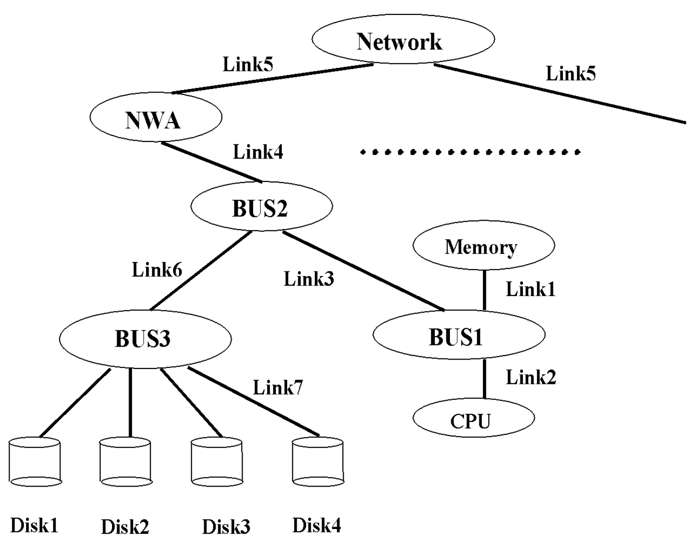

Fig.4: Typical SP2 configuration used in fig. 5 showing the nodeset and linkset components

Now considering the execution script for an application, PetaSIM can either read the execution script from a file or dynamically read it from a pipe connecting to say a University of Maryland application emulator. We can either read all the data at once or do it in block or streaming fashion. The latter has some obvious difficulties with the data dependencies described above in Sec. 3.3.4.

4. Examples and Experimental Results

In this section we summarize some preliminary results from PetaSIM with 3 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[11].

The results 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 on SP2 machine. We have looked a few different machine configurations and in fig 4, we illustrate PetaSIM's abstraction of the SP2 system used in the results presented in fig. 5.

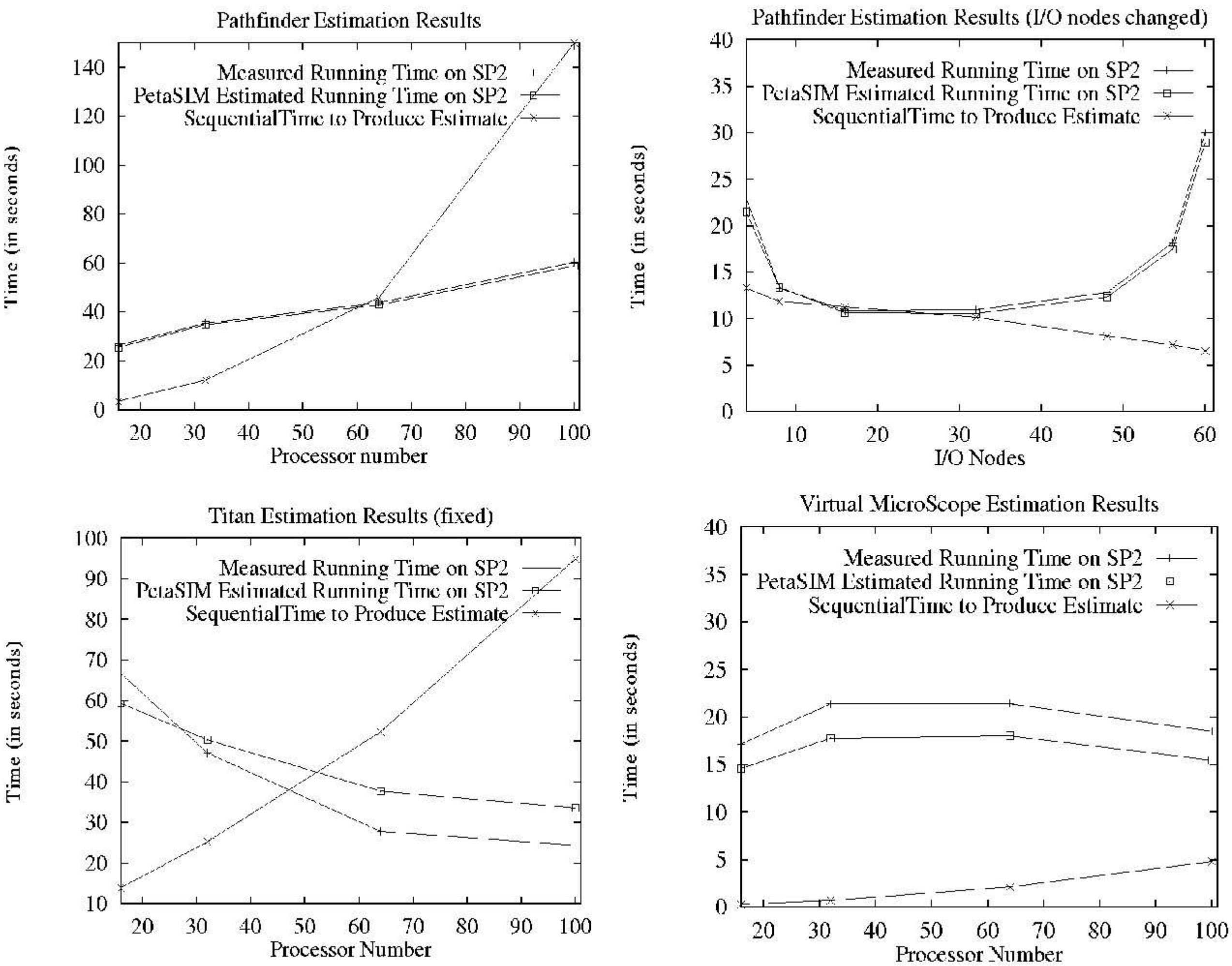

Fig.5: Measured Execution Time compared to the estimated execution time from PetaSIM for four examples. We also show the sequential execution time needed to produce the estimate.

This should illustrate how the *linkset* and *nodeset* objects described in sec. 3 are used in practice. 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 an 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 provide 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. And PetaSIM provide the interface for both inputs from the emulators (like our experience with the University of Maryland's emulators) and from the hand-written code of the system designers, which make it even more flexible.

5. PetaSIM and Distributed Collaborative Engineering

Above we have concentrated on performance estimation for parallel machines although we saw a complex structure when we linked data access and computation. Here we note that PetaSIM's support for hierarchical memories naturally allows its use for computational grids – a geographically distributed collection of people, computers and data resources. We take a particular example a collaborative engineering scenario as is targeted by NASA's Information Power Grid. Here we imagine a geographically distributed set of engineers who are collaborating to design a new vehicle. This would involve interactive linkage to planning and design tools such as CATIA, web and database information resources, multi-component (structures, acoustics, airflow) simulations and their visualization. Both the engineers and the computational components are geographically distributed with a dynamically heterogeneous network linkage. Superficially one can handle this by mapping the computing and information resources into nodesets and the complex network into a suitable hierarchical dynamic collection of linksets and nodesets. PetaSIM's support for rapid prototyping translates into a fast estimation capability, which can be used for adaptive resource management. Again one does not need very precise estimates as such dynamic systems are inevitably specified imprecisely. There will be some straightforward extensions of PetaSIM to handle different geometrical distribution and for its use in a feedback loop as part of a control system. However there are also some qualitatively new circumstances, which we now discuss.

The linkset performance estimator would be linked to the emerging set of low level network quality of service tools. In this regard, PetaSIM would be regarded as the application level quality of service interface. The collaboration (shared visualizations, information and standard tools such as audio-video conferencing and whiteboards) can be handled by systems like Habanero or TangoInteractive [12,13]. However there is an opportunity to link PetaSIM with these collaborative systems in non-trivial fashions. For example, these shared event collaboration environments typically allow different views of a particular shared object for each client. In this regard, information from PetaSIM could be used by the collaborative system to decide on the frame-rate and codec (MPEG, H263 etc.) to be used in multimedia streams. PetaSIM can be used in this tactical fashion, for TangoInteractive to determine preferred audio, video or image format for each client. Alternatively, one can use it in strategically in the design of the network and associated information servers (nodesets). As a particular example, consider a shared browser (providing a shared web page) which is at the heart of TangoInteractive's support of distance education. In this case it provides the curricula information provided by the teacher to the students. However in collaborative engineering, this would be a briefing, the current corporate time-lines, an important memo or similar information to be shared and discussed. In the case of the Web, we are used to asynchronously delivery of information in this fashion. However this shared application has to be delivered at the same time to within seconds at all clients so it can indeed be discussed synchronously with the linked audio-video and other resources. The systems engineer would address this constraint by either providing appropriate (possibly multi-cast) network bandwidth, by use of mirror or proxy web servers or dynamically by adjusting the resolution used for the images sent to individual clients. Note that a high quality shared image or visualization requires substantially more bandwidth than the typical low bit rate video needed for "talking heads" video-conferencing. In either case the value of a performance estimator such as a PetaSIM is clear and further one would need significant extensions in its capabilities to describe the new data streams and different goodness measures.

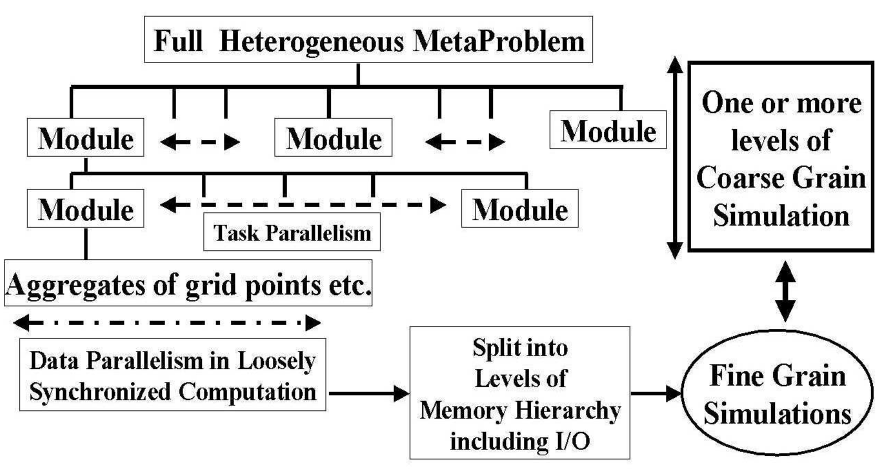

Fig. 6: Support for coarse grain modules in PetaSIM

Now let us turn to the computational component of this collaboration. Here we have supposed a set of geographically distributed simulation and data modules linked together as a meta-problem. These could be supported by systems such as NPAC's WebFlow [14] and often would be linked in some dynamic dataflow paradigm. This is familiar in the image-processing arena where the system Khoros [15] is often used to coordinate such modules. In these examples, modules are relatively large and coarse grain. Often each module is internally data-parallel and could be running on a tightly coupled parallel machine. PetaSIM has been designed from the start to support a mix of coarse-grain functional (dataflow) of data parallelism as shown in fig.6. However we have not yet implemented or tested all these new capabilities. The computational part of a collaborative session would be used both asynchronously and synchronously. In the first case, a large scale multi-module metacomputing application would be run off line producing results which could be part of the shared information resource used in a collaborative session. More ambitiously, perhaps the linked computational modules would be running during the collaborative session to update a planning tool or drive a shared visualization. PetaSIM could support either type of activity.

6. Related Projects 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. [10]

Compared with some other performance estimators, PetaSIM has some special characteristics. Most of the other simulators, such as the University of Maryland’s systems [9], [11], 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. In some cases PetaSIM’s interpreted processing of the ASCII *execution script* [3], may be too slow and some pre-processing (compilation) of the execution script should be added as an option. Remember one of our goals was to support the study of a range of architectures for a fixed application suite and in this case, it makes sense to compile the execution script.

Other extensions of PetaSIM include execution of the estimate in parallel and perhaps more interestingly, adding capability to address the novel types of problems sketched in sec. 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] Geoffrey C. Fox, Roy D. Williams, and Paul C. Messina, Morgan Kaufmann, "Parallel Computing Works!”, 1994

[3] 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.

[4] 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/

[5] M. Rosenblum, S. Herrod, E. Witchel, A. Gupta, "Complete Computer System Simulation: The SimOS Approach", IEEE Parallel and Distributed Technology: Systems and Applications, 3(4), winter, 1995, pp 34-43.

[6] E. Brewer, A. Colbrook, C. Dellarocas, W. Weihl, "Proteus: A High-Performance Parallel Arhictecture Simulator", Performance Evaluation Review, 20(1) Jun 1992, pp 247-8.

[7] D. Park, R. Saavedra, "Trojan: A High-Performance Simulator for Shared Memory Archtectures", Proceedings of the 29th Annual Simulation Symposium, April 1996, pp 44-53.

[8] J. Veenstra, R. Fowler, "MINT: A Front-end for Efficient Simulation of Shared-Memory Multiprocessors", Proceedings of International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Feb 1994, pp 201-7.

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

[10] 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

[11] 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

[12] Tango Collaboration System, developed at NPAC mainly for distance education and online at http://trurl.npac.syr.edu/tango

[13] Habanero Collaboration System, pure Java collaboration system developed at NCSA at online at http://www.ncsa.uiuc.edu/SDG/Software/Habanero

[14] E. Akarsu, G. Fox, W. Furmanski and T. Haupt, "WebFlow - High-Level Programming Environment and Visual Authoring Toolkit for High Performance Distributed Computing", paper submitted for Supercomputing 98, http://www.npac.syr.edu/users/haupt/ALLIANCE/sc98.html

[15] Khoros Integration Software from Khoral Research and online at http://www.khoral.com/