Petasim V0.3.3 Report

Northeast Parallel Architectures Center
Syracuse University
111 College Place
Syracuse, NY 13244–4100

July 1, 1998

Abstract

In the design of parallel computer systems and the developing of the applications, it is very important to have some good performance prediction tools to get an estimation result before the real architecture construction and application implementation. This paper describes a new approach – `Petasim` , which will get the abstractions of the applications and the underlying machine architecture to give performance estimation. In this approach of performance estimate, one can easily modify all the architecture parameters of a computer system, and the behavior of the application to get the performance and execution cost very quickly, which will greatly help the new computer architecture and application deign.

1 Introduction

Performance prediction for a large scale application, such as weather forecasting, oil exploring, etc, is a very important and rather complex problem for scientific computing, especially when involing the massively parallel computers. Provided with a good performance estimator, one can easily construct the application model, adjust the system parameters to find out a suitable computing model. We introduce a new approach in our design of `Petasim`, which will benefit both the computer system design and the application design.

Performance estimator *Petasim* is built around the performance prediction process sketched in Fig.**??** [**?**]. 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 heart of this performance prediction process are two technologies - HLAM (Hierarchical Application Modeling Framework) and Petasim [**?**], [**?**]. In this report, we will address the design of *Petasim*, which

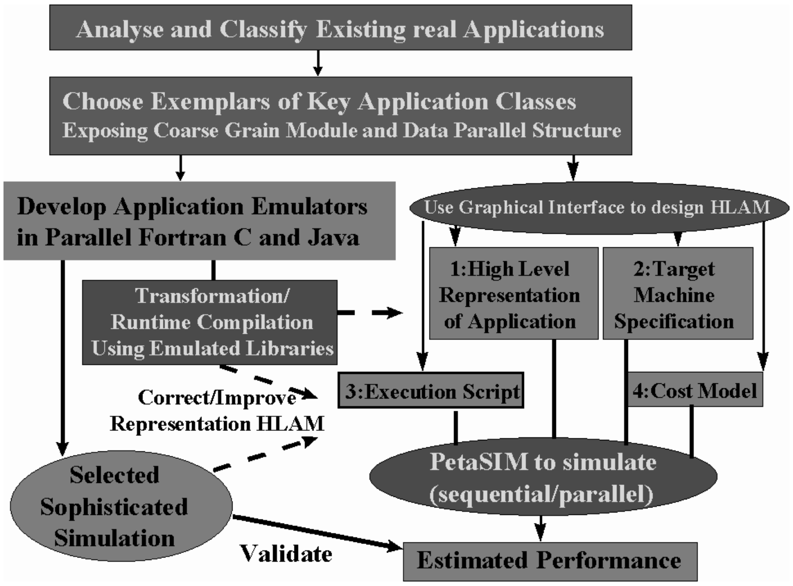

Figure 1: The Performance Prediction Process

will take three key inputs from HLAM, 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.

2 Description of Petasim

Performance estimator PETASIM is aimed at supporting the (conceptual and detailed) design phases of parallel algorithms, systems software and hardware architecture. Originally this was designed as a result of the two week long workshops - PAWS and PetaSoft - aimed at understanding hardware and software architectures ten years from now when Petaflop (10^{15}) scale computing can be expected. It was clear from these meetings that the community needed better aids for performance estimation as the 8 groups (and 8 different machine designs) present found it difficult to compare designs and in particular differed by a factor of million (PAWS report) in estimating performance of even a set of extremely simple algorithms - the PetaKernels – on their new designs [?], [?]. The most sophisticated PetaKernel was a regular finite difference problem solved by simple Jacobi iteration which is of course well understood. 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, adaptivity and multiple program components which make even high level performance estimates hard. It uses a crucial simplification - dealing with data in the natural blocks (called aggregates in HLAM) 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. Then we will have:

Speed Up = Number of Nodes/(1 + Overhead)

where the Overhead is proportional to $(GrainSize)^{-g}(t_{comm}/t_{calc})$

where in this case 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 natural data block 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, 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 HLAM and 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.

Fortunately we have good evidence that we can generalize this naive analysis to more complex problems and machines for indeed the Rutgers/UCSB group has studied granularity issues {cluster,dsc} to identify the natural data block sizes and computation clusters based on computation/communication ratios in more general hierarchical memories. They have developed preliminary performance prediction and optimization tools (PYRROS {PYRROS}, D-PYRROS {jjiao},RAPID {RAPID}) based on task graphs in which the impact of single-processor memory hierarchy is addressed in the intra-task level and the impact of inter-processor communication delay is identified in the inter-task level. These techniques have been shown effective for a number of adaptive and static applications {Iter-2,ship,Fernandez,SparseLU} and it will be naturally integrated into PETASIM as this Rutgers/UCSB technology is the basis of HLAM.

As well as a machine description, PETASIM requires a problem description where we will use the HLAM as described above. Note also that application emulators also use the same "aggregate" description of applications needed by PETASIM and these will be used as input. It is not clear yet if we will convert these directly to the HLAM or allow a separate API for this style of program description. We will provide even more work here later.

3 The Design of Petasim

3.1 The 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, linkset* while the applications can be specified using *dataset, 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.

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

2. ***linkset* Object Structure in PetaSim Estimator**

   A basic *linkset* has the following properties. A derived linkset is gotten by concatenating several basic *linksets* 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*

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*

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 done using ghost cells so that edges of regions transmitted.

5. **Computation/Communication Instructions (Execution Script)**

   Much of 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 is stopped by this. Otherwise it is assumed

   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. While move command will just move the data from source to destination, which does not have that "loop" operation.

3.2 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 HLAM (Hierarchical Application Modeling Framework), which was taken as input in PetaSim performance estimator. University of Maryland has some kinds of emulators right now, which take applications and generate the abstractions of the applications. After the initial co-operation with UMD's emulators, we found that in order to make petasim available for the design of HLAM, we need to provide the *Petasim* with the following features:

1. ***nodeset* member identical**

   In the current version of PetaSim right now, PetaSim has the ability to describe the architecture's specifications into *nodeset* and *linkset*, according the Target Machine's memory hierarchy. In order to describe detail architecture information of the machine, we also need to describe each nodeset member's feature and its linkage features in the architecture description files. Meanwhile, in the execution script, performance estimation also wants to estimate the cost of each nodeset member. So we provide the nodeset member identical in the architecture and application description files. In this case, we can estimate the performance cost on each nodeset member.

2. **dataset hierarchy**

   In this version of PetaSim, an application may contain a set of dataset, each contains *totalsize* units of data. In the assumption of PetaSim, all computation is supposed to be in block-block form with both:

   - "real" blocks of nonzero size and
   - the "virtual zero size blocks formed by for instance ghost cells around the edge of domain

   The blocks are formed from *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, like the applications from UMD, the applications may have the feature of hierarchy. An application contains a set of *dataset*, and a dataset may contain a bunch of *sub-dataset*. A statement in execution script may operate on a specific *sub-dataset* or directly operate on a parent dataset. For example:

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

   And the execution script may be like :

   ```
   move sub_data1 from memory_node to cpu_node
   compute sub_data1 on cpu_node ; which works on sub_dataset
   move dataset1 from disk_node to cache_node
   mode dataset1 from cache_node to memory_node
   ; which automaticly operates on sub_data1, sub_data2,
     subdata3 and sub_data4
   ```

   We also provide this feature of dataset hierarchy in the current version of Petasim. This will benefit the application's HLAM to generate easy execution script, and make it more readable.

3. **execution script library**

   Right now, in the *execution script* of PetaSim, we support four different kinds of statement, *send, move, Use distribution, compute* and *synchronize* commands, which is enough to express the abstractions of operations in the applications, data movement and computation. But this simplicity of abstraction somehow will make the execution script of applications very long, especially to the real large applications. So we should build an execution script library, which contains aggregates of data movement and computation in a function call. For example:

Calculate dataset on CPU_nodeset

will involve the following operations :

```
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.

4. **data dependency**

   In the current version of PetaSim, the performance estimation of 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. And the estimation operation is just in the order the statements of the execution script. The petasim will not try to reorder the statements, like the real application execution.

   But in the real case, some operations may not have the order of statements in the execution script. They will depend on the finish of previous statements. For examples:

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

   In this example, s3, s4 may not the execution order in real execution. They just depend on the finish of s1 and s2. If s1 finish first, s3 may execute first on node2, and if s2 finish first, s4 may execute before s3. They just have the data dependency with the previous statement.

   Right now, Petasim doesn't have the mechanism to deal with this kind of data dependency problem. Any way, we will try to support that later.

3.3 Interface Petasim with HLAM

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. In order to provide *Petasim* use for HLAM design, and try to avoid the large amount of execution script file, we may use socket communication between the Petasim and HLAM.

In the architecture Description of Machines, we have two parts, nodeset and linkset, which aimed to describe the detailed memory hierarchy and the linkage relationship between each hierarchy level. In the nodeset description file, it contains all the information about each nodeset, and all of its *nodeset members*, and the linkage neighbors of this *nodeset member*. While in the linkset file, it contains the features of all the linkset, and each linkset member, which link two of the nodeset members.

In the aspect of execution script of an application, PetaSim will read the execution script from a file. And it supposes that the total statements of execution script have already existed there. In order to use the socket format to work with HLAM, we need to take execution script input dynamicly, and give an estimation. Which means, from the HLAM, it will dynamicly generate the execution script statements according to the applications and then send to petasim to give an estimation. This is also a way to try to avoid the long execution script of an application.

Here, we plan to send and receive execution script statements in a block way, which means, it supposes that each bunch of execution script statements petasim receive from emulator will have a rather independent execution behavior. Petasim doesn't need to care about the data dependency of the statements in this set with the execution script received from emulator previous or later. Petasim will estimate the cost according to the hardware available now and the execution script received this time.

4 Further Work

As discussed above, the early experience of *Petasim* prototype shows that if we can get the correct abstraction of the machine architecture and the execution behavior of the applications, *Petasim* can produce a performance prediction result in a very short time. This will make great benefit to the new architecture design and the application development.

The early experience also shows that in order to make *Petasim* more usable to the design of HLAM, we may also consider the binary expression in the architecture and application description. Right now, we take the ASCII expression in all the description files, which may make them very large files, especially in the massively parallel computing. The binary code expression will make them in a short size. However, in order not to loose the flexibility of *Petasim* we may also provide some kind of translator between the ASCII user interface and the binary code expression. We may provide a much attractive programming environment based on our early experience [**?**].

Typically *Petasim* is executed on a sequential machine as it is aimed at relatively crude estimation. However if for instance an accurate modeling of cache, requires a small aggregate size (and corresponding large number of aggregates), the performance estimate may become very expensive computationally. Very adaptive problems could lead to a similar situation. In this case, we may need to use parallel estimation and the operation of *Petasim* needs to be parallelized.

We are also considering to improve *Petasim* to estimate the program's execution even more detail to make it an application simulator. If it works in a parallel way, we can dynamically simulate the execution behavior of an application on a sequential / parallel machine.

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 Chapter7).

[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 Paralle Processing Symposium, April 1997.