

White Paper on Proposed Research

Organization: HCS Research Laboratory, University of Florida
Principal Investigator: Dr. Alan D. George, Associate Professor of ECE
Project Title: Performance Analysis Tools for Shared-Memory HPC
(Phase 1: Research Study and Comprehensive High-Level Design)

1 Introduction

Complex programs are often required to solve complex problems. As the intricacies of the problem increase, so does the developer's dependency on performance analysis and optimization tools to help identify and address performance bottlenecks and thereby achieve the desired outcome. Meanwhile, system architectures for high-performance computing (HPC) continue to become increasingly complex in both breadth and depth. Unfortunately, as HPC architectures and programming models have evolved, potent tools for performance analysis have often not kept pace. In many cases, such tools have been an afterthought in the view of commercial HPC vendors. The negligence by vendors in designing, standardizing, and refining such tools has limited support for developers to only a few HPC system architectures or a limited span of programming models. For example, in recent years MPI has dominated the HPC programming world, and therefore has received the lion's share of attention on research and development toward performance tools. Other programming methods and languages, particularly ones for shared-memory parallel computing such as SHMEM and UPC, are becoming increasingly important in the HPC community but are desperately lacking in powerful and versatile performance tools.

2 Scope

This document proposes the first phase of a research project that ultimately will address the aforementioned challenges. In this first phase, researchers at the University of Florida will conduct basic and applied research to investigate key concepts and develop a comprehensive high-level design for a performance analysis tool (PAT) or suite of such tools that will directly support performance analysis and optimization of UPC and SHMEM programs on complex HPC architectures and systems. This effort will study existing and emerging performance analysis theory and tools, current and future HPC architectures, and the UPC and SHMEM programming languages. The results of the first phase will set the stage for a second stage where the comprehensive high-level design will serve as the basis for implementation, evaluation, and refinement of the PAT for UPC and SHMEM computing.

This first phase of the research project will encompass several tasks over the course of one year. During this phase, researchers will conduct an evaluation of the strengths and weaknesses of available tools for performance analysis and target architectures that support UPC or SHMEM. In parallel, the UPC and SHMEM programming models will be studied to determine the types of performance information that should be captured, analyzed, and used by the PAT. A comprehensive high-level design of a PAT prototype for UPC and SHMEM on selected HPC platforms will be produced at the end of the first phase. The structure of specific tasks for the follow-on or second phase will not be fully developed until the end of the first phase. General tasks

in the second phase are expected to include implementation, functional testing, performance testing, usability testing, and design refinements of the prototype PAT, as well as further investigations to improve performance-gathering techniques in terms of both efficiency and effectiveness.

3 Background

This section provides brief background information on issues associated with the proposed project. The topics of this background include HPC architectures, UPC, SHMEM, and existing performance tools.

3.1 HPC Architectures

High-performance computing aims to solve large problems that cannot be solved in a reasonable amount of time using a conventional computer system. HPC architectures offer a large amount of processing power in a controlled environment. Various processor designs, such as RISC, vector, DSP, and VLIW, as well as related new technologies in reconfigurable computing, et al. may be employed to exploit specific aspects of HPC needs. A variety of memory hierarchy designs, with options from various categories of either shared memory or distributed memory, are used in conjunction with processor designs alongside a wide array of communication network and synchronization options in hardware. In addition to system and subsystem architecture, also very important in the performance of these HPC systems are the operating systems used to manage the resources, middleware used to exploit the resources more conveniently and effectively, and compilers to target processors and other resources. HPC systems are among the most complex systems in the digital world. New systems are becoming increasingly complex and heterogeneous, leveraging new concepts and technologies and doing so at a greater number of levels, from chip multiprocessors to grids.

A number of existing and emerging systems are of potential interest as targets for this project. A preliminary survey of HPC systems could include such machines as SGI's Altix, HP's 9000 and AlphaServer, Cray's X1 and OctigaBay 12K, IBM's pSeries and SP, as well as a variety of high-performance clusters connected by various system-area networks such as InfiniBand, Myrinet, Quadrics, or SCI.

An important part of the controlled environment provided by HPC systems can be the availability of mechanisms for resource monitoring found in the environment. A closer look into many HPC systems shows that some systems can provide low-level mechanisms for monitoring system state and performance. Oftentimes vendors may use such mechanisms for system development but disable them when machines are prepared for delivery to end-users. However, these mechanisms could contribute significantly to tools for performance analysis. Thus, it is important to investigate and interact with key vendors about such mechanisms and the possibility of gaining access to the mechanisms. For example, a preliminary investigation by our team with SGI has shown that their new Altix family of scalable shared-memory multiprocessors possesses many such low-level mechanisms that are disabled in delivered machines but could be enabled again and in so doing support performance analysis tools. Our preliminary investigation has yielded an offer by SGI to provide documentation and access to these mechanisms toward the development of our PAT.

3.2 SHMEM

Cray Research originally created the Shared Memory (SHMEM) library for use on the T3D supercomputer. The SHMEM library provides a shared-memory model for programming of parallel computers. It allows a processing node to read and write data stored in the physical memory on another processing node. In addition to the basic set of primitives for accessing remote memory (e.g. *get* and *put*), the SHMEM library provides collective parallel operations. Though largely available on Cray and SGI systems, SHMEM is also available for numerous other HPC machines, including machines equipped with either QsNet from Quadrics or Scalable Coherent Interface from Dolphin and its Scali Wulfskit Software Platform (SSP) for interprocessor communication. Emerging systems also tout support for SHMEM, such as the OctigaBay 12K machine.

Portable support for SHMEM middleware exists in several forms. GPSHMEM is an effort from Ames Laboratory to increase the portability of SHMEM by implementing SHMEM function calls using the Aggregate Remote Memory Copy Interface (ARMCI), a portable remote memory copy library, coupled with MPI libraries. Additionally, future developments of the GASNet communication library at UC Berkeley will include support for SHMEM on a wider set of platforms via a shared-memory model provided by GASNet on top of a variety of architectures [1].

3.3 UPC

Unified Parallel C (UPC) began to take shape in the late 1990s, and Version 1.0 of the language specification became available in February 2001. Since then, UPC has caught the attention of many organizations in educational institutes, government, and industry. The key to its success is the ability to create powerful shared-memory parallel programs with a C-like syntax. Its ability to use a shared-memory model on a variety of architectures is making it very important in a number of high-performance applications.

UPC is available on a growing number of HPC systems. Examples of newer platforms with UPC support include the HP 9000 and AlphaServer, Cray X1, and SGI Altix. In addition, IBM SP machines, as well as any systems with MPI, and a broad range of cluster systems connected by Quadrics, InfiniBand, and Myrinet, can all support UPC albeit with varying degrees of performance via the Berkeley UPC runtime system [2]. Recent additions of UPC include tools that target the full family of HP systems based on the HP-UX operating system and PA-RISC CPUs, as well as computing clusters that feature SCI system-area networks (i.e. a UPC/GASNet conduit is under development in our lab this year via collaboration with the UPC group at UC Berkeley).

3.4 Existing Performance Tools

There are numerous performance tools available with support for sequential and parallel computing and programming languages. Sequential C and MPI profiling tools are the most common, with some other tools offering support for shared-memory programming models, including OpenMP and SHMEM [3]. Many performance tools are based on the popular Performance Application Programming Interface (PAPI), which provides access to hardware counters on a variety of

platforms that can be used to track a wide range of low-level performance statistics such as number of instructions issued, L1 cache misses, and data TLB misses [4-5]. As hardware support must be present on a particular processor to make measurements of this type, the set of available statistics to monitor is dependent on the architecture of the processor executing the code. Many tools such as SvPablo successfully leverage PAPI in order to provide low-level hardware measurements while also providing other higher-level measurements such as function call durations or synchronization overhead [6-7]. In addition, some tools such as Kojak also attempt to precisely identify performance bottlenecks and qualify them according to performance impact. While very few tools support SHMEM or UPC, for those that do the support is typically limited to a particular architecture (e.g. CrayPat) or provide only minimal support (e.g. Vampirtrace only supports function call instrumentation for SHMEM). To our knowledge, none of the existing tools is able to provide direct guidance to users on how to improve program performance, and thus this task is still relegated almost completely to users. Table I provides a brief summary of key tools currently under consideration, with more to be added as they may emerge.

Table I. Overview of Existing Performance Tools

Tool	Tool Developer	Programming Models	Platforms	Features
CrayPat	Cray	MPI, Pthreads, SHMEM, UPC	Cray systems	Re-building/re-linking required, uses hardware performance counters
Dimemas	European Center for Parallelism of Barcelona (Spain)	MPI, PVM, PARMACS	IRIX, Tru64, AIX, HP-UX, Solaris, Linux	Tracks CPU usage of code blocks, traces analyzed using Paraver and Vampir
Kojak	University of Tennessee / Forschungszentrum Jülich (Germany)	MPI, OpenMP	Linux, AIX, IRIX, Solaris	Automatic instrumentation, automatic analysis of trace files, identify performance problems
MPE and Jumpshot	Argonne National Laboratory	MPI	Linux, Unix, Windows	Automatic instrumentation of MPI programs with MPE, visualization with Jumpshot, included with MPICH
PAPI	University of Tennessee	N/A	Windows, Linux, Unix	High- and low-level interface to hardware performance counters, used by many tools
Paradyn	University of Wisconsin	MPI	Solaris, Linux, Windows, AIX	No mods to source or binary, dynamic instrumentation, identify performance bottlenecks
Perfometer	University of Tennessee	MPI	Any with PAPI and Java	Monitor local/remote apps at runtime, alarms to pause program at thresholds
SvPablo	University of Illinois	MPI, OpenMP	Linux, Solaris, IRIX, AIX, Tru64	Interactive instrumentation, library linking, correlates performance data with source code
TAU	University of Oregon	MPI	Linux, IRIX, AIX, Windows, Solaris, UNICOS,	Performance data for threads, 3 different instrumentation methods
Vampirtrace	Pallas	MPI, Global Array, SHMEM	Linux, Tru64, UNICOS, Solaris, AIX, IRIX	Multithread MPI support, library linking, record arbitrary user-defined events

4 Approach

To gain an in-depth understanding of issues relevant to program performance, we propose five layers for the framework of research focus, as shown in Figure 1. These layers help divide the workload involved in studying programming models and HPC architectures and the performance analysis issues and tools related to them. The application layer deals with general parallel programming issues, as well as issues unique to the problem at hand. The language layer involves issues specific to the UPC or SHMEM programming model. The compiler layer includes the effects of different compilers and their optimization techniques on performance. The middleware layer includes all system software that relates to system resources, such as the communication protocol stack, operating system, runtime system, etc. Finally, the hardware layer comprises key issues within system resources such as CPU architecture, memory hierarchy, communication and synchronization network, etc.

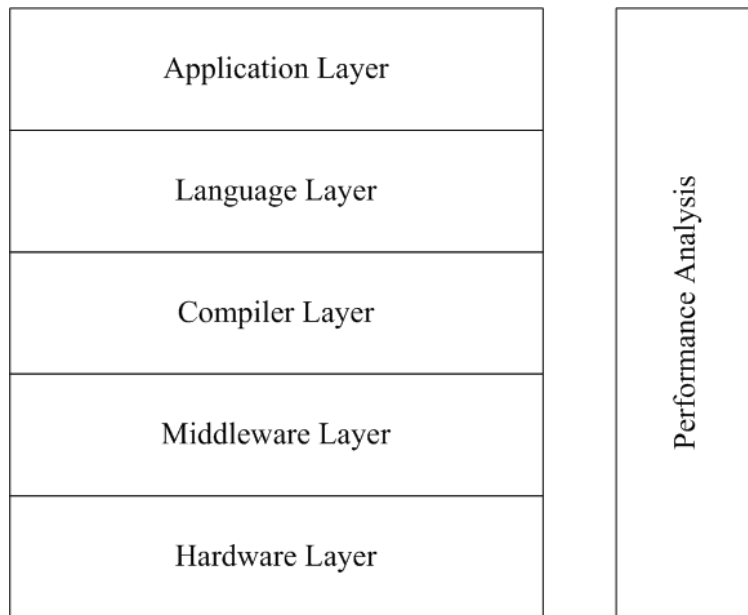


Figure 1. Layers of Performance Analysis

4.1 Research Strategies

The following sections describe two different strategies that can be used to develop a high-level design for a PAT prototype. A comparison between the two is given and a hybrid strategy is presented to capture the advantages of both.

4.1.1 Tool-Driven (Bottom-Up) Strategy

The Tool-Driven strategy shown in Figure 2 starts with the study of existing tools for specific HPC architectures to obtain a list of measurable factors (any characteristic that affects the program

performance when altered). Each factor will then be examined closely through experimental studies to determine its relevance in each of the different layers. Once these relevant factors are identified, extensive experiments will be performed to determine the exact relationship between these factors and program performance (degree of sensitivity, etc.). Finally, factors having significant impact will be incorporated into the PAT design.

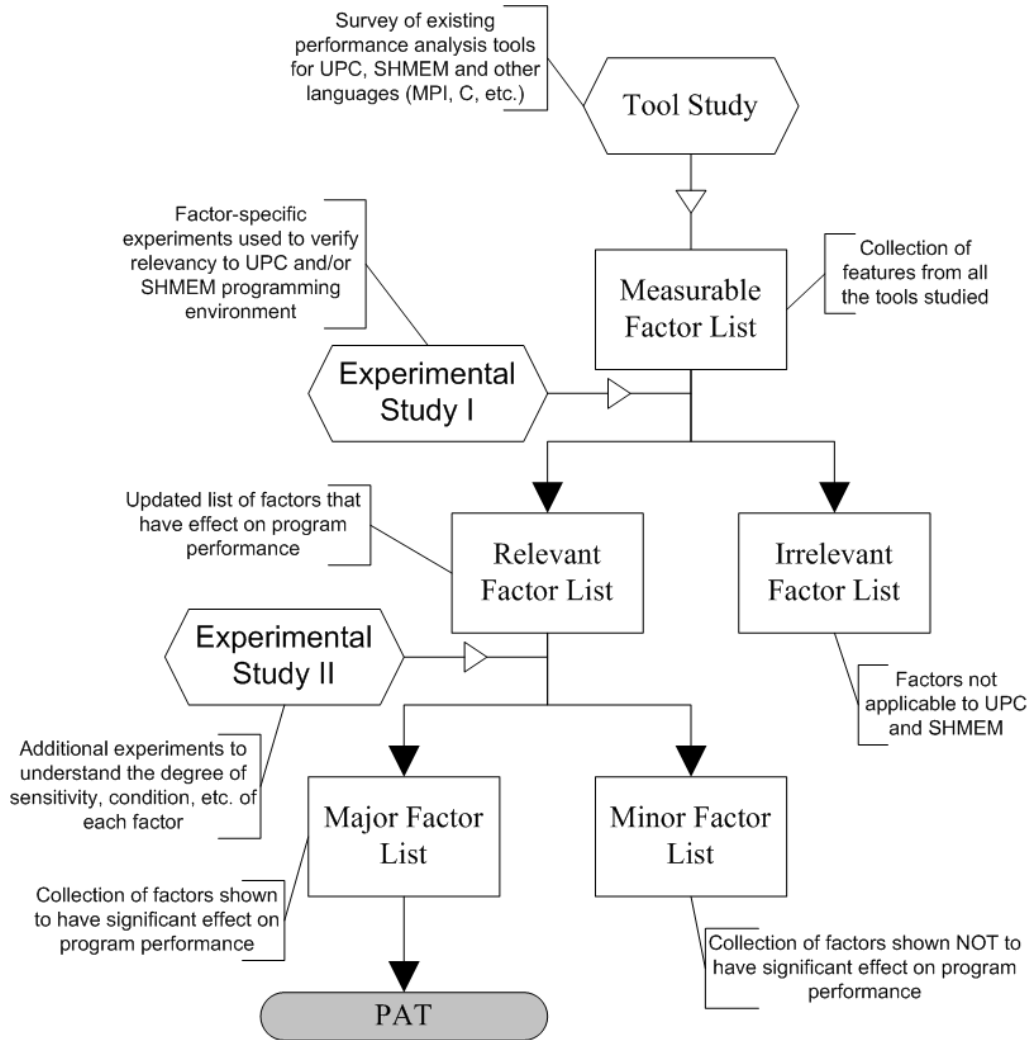


Figure 2. Tool-Driven Strategy

4.1.2 Layer-Driven (Top-Down) Strategy

As illustrated in Figure 3, the Layer-Driven strategy aims to identify all possible factors in each of the five different layers of performance analysis and optimization. Independent investigations in each of the layers are needed to identify layer-specific factors and possible high-level analyses. Factors and analyses from different layers will then be examined and compared to yield a minimal and complete set of factors (i.e. eliminate overlapping factors and produce additional high-level analyses). From this preliminary factors list, experimental studies will be performed to determine the relevant factors that will then be compared with the results from tool study to yield a measurable

factors list (i.e. identify relevant factors that can be measured). Finally, extensive experiments will be conducted to determine the factors that will be incorporated into the design for the PAT.

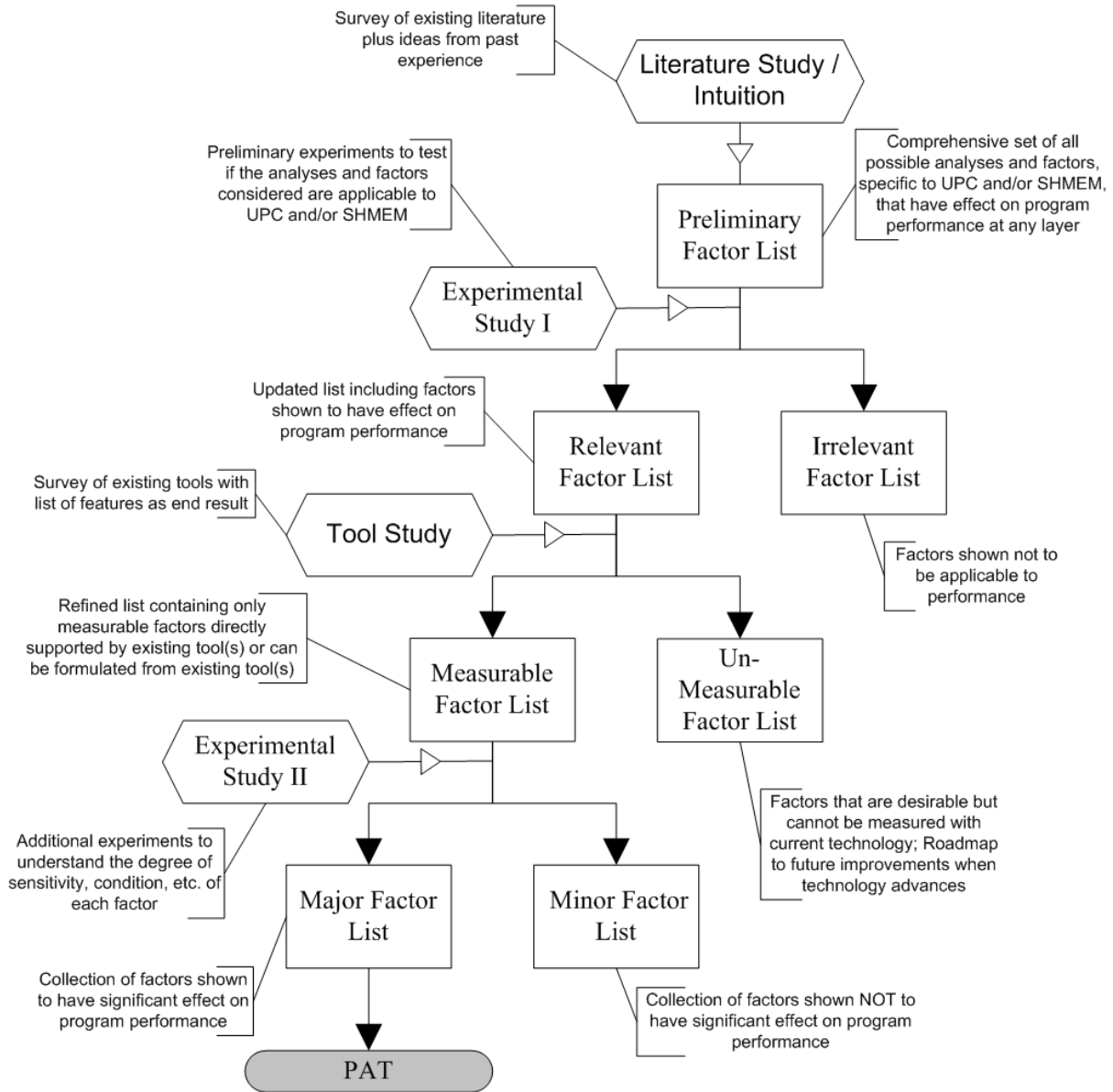


Figure 3. Layer-Driven Strategy

4.1.3 Strategy Comparison and Combination

With the Tool-Driven strategy, the starting factor list is easier to obtain since it is the collection of the key features from existing performance analysis tools. However, due to its bottom-up nature, it is more difficult to combine these factors into high-level measurement and analyses. By contrast, the Layer-Driven strategy facilitates this task more readily because high-level analyses are generally formed before individual factors can be identified (which is the nature of the top-down strategy). In

addition, the factors list obtained through this process is more comprehensive and thus may prove to be more useful to end users of the PAT. Table II summarizes the advantages and disadvantages of each strategy.

Table II. Strategy Comparison

	<i>Tool-Driven</i>	<i>Layer-Driven</i>
<i>Starting factors list</i>	Easier to obtain	More comprehensive
<i>High-level analysis</i>	More difficult	Easier
<i>Development time</i>	Shorter	Longer

In order to minimize the time of development and maximize the usefulness of the PAT, we propose a hybrid strategy as illustrated in Figure 4, where the first step involves studying of the layers and tools in parallel to obtain a relevant, measurable factors list. This list will serve to determine the final set of factors to incorporate into the PAT.

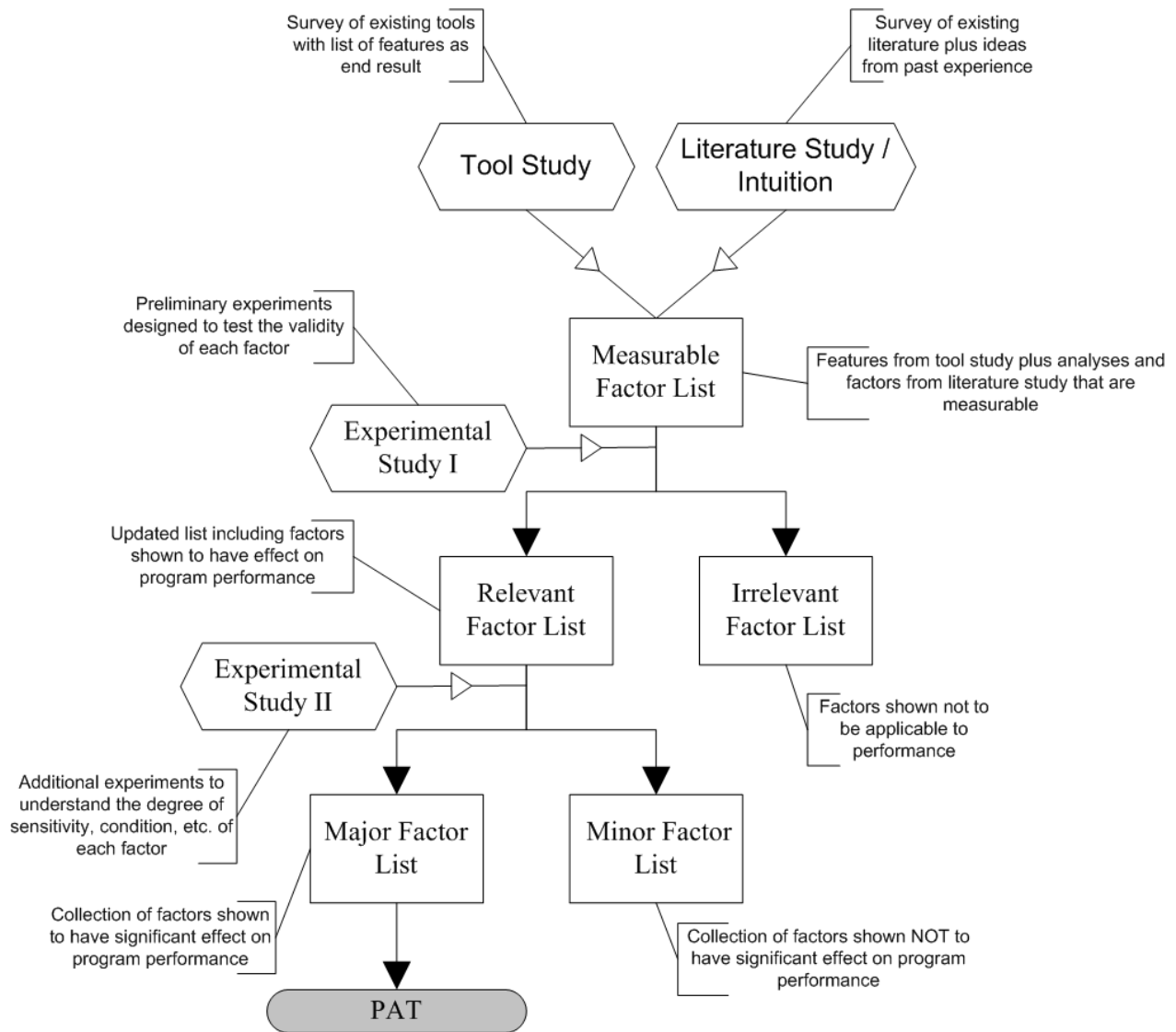


Figure 4. Hybrid Strategy

4.2 Tool Study

This project will incorporate an in-depth analysis of the existing tools that were introduced in Section 3.4. The analysis will require a significant degree of knowledge and experience with each tool in order to become closely familiar with its capabilities and limitations. Those tools without direct support for either UPC or SHMEM will be judged based on performance collection strategies so that enhancements and/or integration with other tools are thoroughly explored.

Once each tool has been scrutinized and assessed according to a rigid quantitative review, the highest scoring tool(s) will be selected. Problem areas will be identified for the chosen performance tools and a plan will be devised to address them. The goal of the tool study is to identify the tool(s) that provide the necessary information or foundation to support UPC and SHMEM programs. Integration points as well as enhancement regions will also be identified as starting points to explore when entering the prototype design phase.

4.3 Layer Study

Parallel to the tool study, a detailed investigation of the factors relevant in the various layers will be conducted. For the application and language layers, study of existing literature and experience with the programming models (e.g. writing programs in UPC and SHMEM) will yield an initial list of factors and provide background for formulating a survey for software developers. The survey will consist of questions aimed to identify factors deemed significant. The results gathered from the survey will then be used to refine the initial list. To identify the factors for the compiler, middleware and hardware layers, vendors and developers will be contacted to gain an in-depth understanding of all related systems (compilers, runtime systems, etc). Once factors are identified, their relevance will then be verified through an experimental study. An understanding of how compiler and code relate to the performance of platform will also be investigated.

Existing HPC architectures will be evaluated fully from a developer's viewpoint through experiments while using available literature as a foundation. HPC vendors will be contacted to determine the low-level mechanisms available on their respective systems. A set of new and emerging machines from leading HPC vendors (plus clusters) and set of machines that currently do or in the future will likely support UPC and/or SHMEM will serve as a starting point in the HPC architecture investigation. These may include Cray X1, Red Storm, and OctigaBay 12K, HP SP4 and 9000, IBM pSeries, and high-performance clusters with UPC/GASNet over various system-area networks (SANs). Existing architectures will be candidates for targeted architectures of the PAT, whereas the new and emerging systems will serve as the foundation for future PAT design upgrades.

The goal of the layer study is to identify possible factors that should be taken into consideration when evaluating the performance of a program and to identify methods in acquiring measurements for them. These factors will then be evaluated with the tool study results to produce a list of measurable factors for the PAT. A preliminary list is provided in Appendix A.

4.4 Prototype Design

From the results of the layer and tool studies, the set of features and analysis framework for the PAT will be determined. Two types of outputs (both simple and higher level analyses composed from simple measurements) will be provided by the PAT. In performance prediction, pre-gathered information (e.g. network latency, runtime system overhead, etc.) are plugged into a predefined set of formulas to estimate both the optimal and worst-case baseline performance [8-9]. By comparing the actual performance of a program against these predictions, the user will be able to determine if further optimization is useful or not (i.e. optimize only if performance is not close to optimal). The second type is performance profiling and tracing where the actual performance of the program is measured [10-12]. Figure 5 diagrams the overall process of the PAT design.

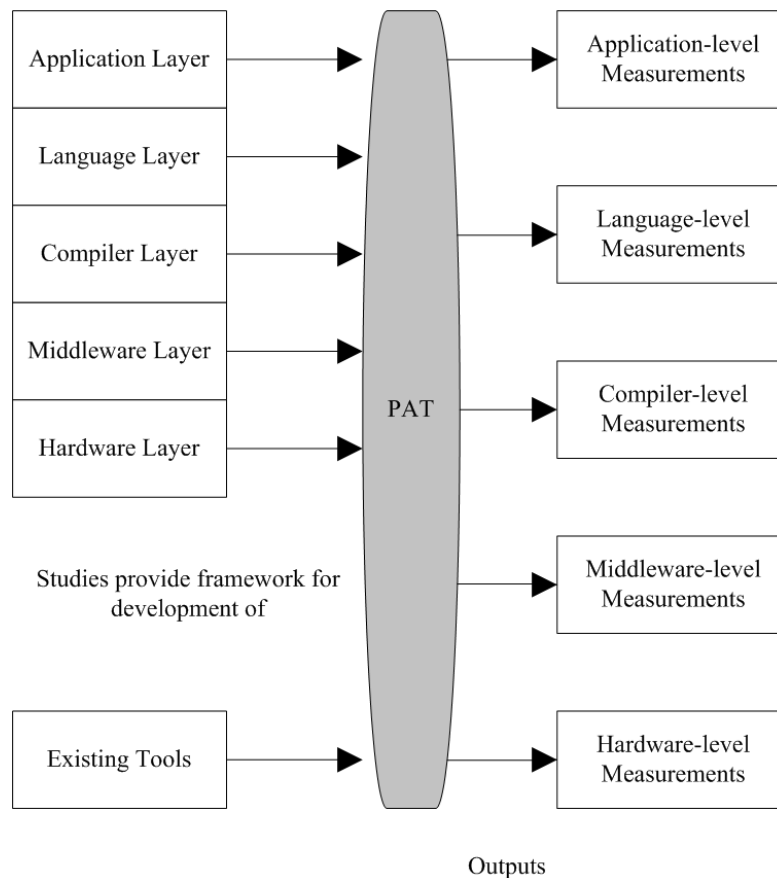


Figure 5. Framework of a PAT Design

Several options exist for the design and eventual implementation of the PAT:

1. Modify existing tool(s)
2. Develop wrappers for existing tool(s)
3. Encourage and enable tool vendor support
4. Develop new tool

Options 1 and 2 are more attractive if possible and practical, since it might require less effort to extend existing tools (e.g. for MPI) to fully support UPC and SHMEM. In addition, the user

learning curve for the extended tools might be shorter for those users already familiar with the original tools. However, depending upon the outcome from this first phase of the project, it may be determined that such an approach is unfeasible due to restrictions concomitant with the tools and developers involved. Option 3 is also attractive if feasible but of course, much like the previous two cases, it depends heavily on the willingness of vendors to fully support UPC and SHMEM on a wide variety of platforms. By contrast, option 4 provides the greatest flexibility in terms of potential capabilities and target platforms, including the opportunity to carefully select the best concepts and features from the field as they related to shared-memory computing with UPC or SHMEM and optimize with such a focus in mind. However, this approach would require an extensive amount of time and effort. Recommendations on the best choice of options will be made at the end of the first phase of the project, as part of the focus on development of a comprehensive high-level design for a PAT prototype. Recommendations will be largely influenced by results obtained through the hybrid strategy process, surveying of end-users, tools developers and vendors, etc.

5 Schedule and Deliverables

Deliverables for this project are proposed at quarterly intervals. They include:

- Q1: Tool and layer study reports; Preliminary experimental study I plan.
- Q2: Experimental study I report; Preliminary experimental study II plan.
- Q3: Experimental study II report.
- Q4: Prototype design document; Phase 2 plan.

The proposed schedule for the first phase (year) of the project is shown in Figure 6.

ID	Task Name	Duration				
			Q1	Q2	Q3	Q4
1	Tool study	12w	■			
2	Layer study	12w	■			
3	Tool** and layer*** study report generation	1w		★		
4	Experimental study I	12w		■		
5	Experimental study I - report generation	1w		★		
6	Experimental study II	12w			■	
7	Experimental study II – report generation	1w				★
8	High-level design – iteration 1	7w				■
9	High-level design – review	1w				■
10	High-level design – iteration 2	4w				■
11	Final report generation	1w				★
12	Phase 2 plan	2w				★

* Deliverables shown with star symbol

** Task includes investigation of existing performance tools and target HPC architectures

*** Task focuses on performance data appropriate in UPC and SHMEM target machines and environments

Figure 6. Proposed Project Schedule (Phase 1)

6 Budget and Costs

The proposed period of performance for the first phase of this project is one calendar year (e.g. 08/01/04 through 08/01/05). The proposed budget allotment for this first phase is \$150K. These funds will fully support three graduate students, all U.S. citizens, during this period in terms of stipends and tuition, and will partially support the principal investigator. Also included in the budget will be requisite expenses for the project in terms of travel, equipment, supplies, etc. The university indirect cost rate is 45.5% added for all budget items except tuition and equipment (which are exempt). The principal investigator has completed a detailed budget plan, and it is currently pending review and approval by the university administration.

7 Preview of Phase 2 Plan

Implementation, evaluation, and refinement of the working PAT prototype on a target set of HPC architectures will be the primary focus of the second phase of this project. The development process for the second phase will adhere to the detailed plan that will be developed and provided as part of the final first-year deliverable. Suitable software and hardware environments will be configured as quickly as possible to shorten the time needed to begin construction of the first version of the PAT prototype.

After the initial implementation and testing of the prototype is complete with Alpha testing, followed by Beta testing with friendly outside sites, then the usability and productivity of the schemes available in the prototype will be evaluated through field-testing by UPC and SHMEM developers and end-users. Since the goal of the development of the prototype is to create an effective and efficient tool for use by UPC and SHMEM software developers, it is important to include their input in the PAT design and testing. These groups will be essential in providing feedback to drive the design and refinement of the tool in order to make it powerful, useful, and user-friendly. Concurrent with outside field-testing, research will continue in the lab during this phase on additional techniques to improve performance information gathering, analysis, and optimization.

8 References

1. *Official GASNet website*, <http://www.cs.berkeley.edu/~bonachea/gasnet/index.html>.
2. *Official UPC website*, <http://upc.gwu.edu>.
3. S. Moore, D. Cronk, K. London, et al., *Review of Performance Analysis Tools for MPI Parallel Programs*, Computer Science Department, University of Tennessee, 1998.
4. K. London, S. Moore, P. Mucci, K. Seymour, R. Luczak, *The PAPI Cross-Platform Interface to Hardware Performance Counters*, Department of Defense Users' Group Conference Proceedings, Biloxi, Mississippi, June 2001.

5. K. London, et al., *End-user Tools for Application Performance Analysis Using Hardware Counters*, International Conference on Parallel and Distributed Computing Systems, Dallas, Texas, August 2001.
6. *SvPablo: A Graphical Source Code Browser for Performance Tuning and Visualization*, <http://www-pablo.cs.uiuc.edu/Project/SVPablo/SvPabloOverview.htm>.
7. L. DeRose and D. Reed, *SvPablo: A Multi-Language Architecture-Independent Performance Analysis System*, Proceedings of the International Conference on Parallel Processing, Fukushima, Japan, September 1999.
8. T. Fahringer and H.P. Zima, *A Static Parameter based Performance Prediction Tool for Parallel Programs*, International Conference on Supercomputing, Tokyo, Japan, July 1993.
9. M. Kuhnemann, T. Rauber and G. Runger, *Performance Modeling for Task-Parallel Programs*, PowerPoint Presentation from Technische University, Germany.
10. J. Lemeire and E. Dirckx, *Automated Experimental Parallel Performance Analysis*, Extended Abstract for 2nd PACT Symposium, Edegem, Belgium, September 2002.
11. G. Jost, H. Jin, J. Labarta, J. Gimenez and J. Caubet, *Performance Analysis of Multilevel Parallel Applications on Shared Memory Architectures*, NASA International Parallel and Distributed Processing Symposium, Nice, France, April 2003.
12. A.D. Malony and S. Shende, *Performance Technology for Complex Parallel and Distributed Systems*, Proc. Third Austrian-Hungarian Workshop on Distributed and Parallel Systems, Balatonfured, Hungary, September 2000.

Appendix A – Preliminary Factors List

Application Layer

Memory placement schema, system size

Language Layer

UPC shared pointers, SHMEM put and get

Compiler Layer

Optimization techniques, compiler-added overhead

Middleware Layer

Runtime system overhead, OS overhead, thread management, communication latency and throughput

Hardware Layer

CPU utilization, cache and memory size, memory latency and throughput, network latency and throughput, network overhead, congestion, network topology, I/O latency and throughput, I/O rate

Appendix B – Resources of the HCS Research Lab

In order to conduct the various studies and experiments as outlined, the HCS Research Lab has many software and hardware facilities already in place, both in-house and through collaboration. In addition, attempts are underway to acquire additional software and hardware support (e.g. remote access to large machines at national labs) to include as many possibilities as we can in our project.

For UPC, our lab is equipped with a HP/Compaq AlphaServer running the HP/Compaq UPC compiler, as well as several Linux clusters connected with SCI, InfiniBand and Myrinet supported by the Berkeley UPC runtime system. Several of our lab members are already familiar with the UPC programming environment and have experience in UPC programming. In fact, the principal investigator has this year started including UPC as a key option for class lectures and projects in his graduate course on parallel computer architecture.

For SHMEM, we are currently working with the vendor Scali to install the SSP software (which includes support for SHMEM running on SCI-connected clusters) on our dual-Xeon and dual-Opteron Linux clusters. We are also in contact with Ames Lab and are currently attempting to install GPSHMEM version 1.0 on our cluster machines. In addition, a collaboration effort with Michigan Tech allows us access to their T3E machine that is SHMEM-enabled.

As for performance tools, we currently have Kojak, MPE, PAPI, Paradyn, Perfometer, SvPablo, and Vampirtrace running on our cluster machines. Also available for use is a custom instrumentation program developed in our lab to extract computation, communication, and synchronization events for SHMEM and MPI programs. This script generation program parses parallel source code and creates as output a script that can be inserted into powerful CAD tools in our lab for performance modeling to create system simulation nodes that compute, communicate, and synchronize as do the real nodes in a real system when running the code that has been scripted. This program can be extended to support UPC and extract any other information that is viable for instrumentation. The instrumentation program also has the potential to integrate other tools such as PAPI to leverage already existing functionalities.

Finally, we have started contacting HPC vendors and research colleagues with whom we collaborate to investigate the possibility of obtaining access to larger machines for this project and (in the case of vendors) in determining the availability of low-level mechanisms in their systems that could be enabled and exploited by a PAT. SGI, Cray, OctigaBay, HP and IBM are among the vendors that we have contacted or plan to contact in the near future. Several other research labs on our campus are also equipped with HPC machines that we may be able to access, including a small-sized SGI Altix and a medium-sized IBM SP. In addition, we plan to contact research colleagues at several national labs (e.g. ORNL, Sandia) with whom we have recently had interactions to inquire about limited remote access to some of their resources (e.g. Cray X1 and 256p SGI Altix at ORNL, Cray Red Storm at Sandia) in support of this project.