Description: Personalized Turnkey Superclusters (PeTS)

In comparison to traditional supercomputers, systems built by clustering PC hardware are more accessible and offer a significantly wider range of hardware and software configuration options. Since February 1994, when we built the first parallel processing Linux PC cluster, we have been exploring the full range of these system-design choices. Unfortunately, to achieve the full benefits from many of the new technologies that we have been developing, application codes must be radically restructured and the cluster design must be tuned to the application. This proposal seeks to build **Personalized Turnkey Superclusters (PeTS)** that will demonstrate the new technologies by restructuring important scientific or engineering applications and creating optimized system configurations that can be easily replicated.

To its users, a PeTS system will appear as an inexpensive dedicated piece of "laboratory equipment" that directly solves their most important computational problems, providing supercomputer performance for those problems with minimal user effort. The proposed project will design, build, and user-test PeTS systems for one or two scientific or engineering applications each year. These PeTS designs will be cheap to replicate, and the design and all software (packaged as a turnkey distribution) will be widely disseminated, qualitatively improving the effectiveness of scientific research in each field for which a PeTS system is created. The success of these systems will help to establish our new technologies as standard tools that can and should be used in implementing other applications.

The team we have assembled is uniquely qualified to accomplish the proposed work. Our team combines some of the most experienced systems-level PC cluster researchers with the scientific and engineering user base, application support, and administrative resources of the well-established University of Kentucky **Center for Computational Science**.

1. What Is A PeTS System And Whom Will It Help?

Physically, a PeTS system is a computing system built using a combination of COTS (cheaply available Commercial Off-The-Shelf components) and openly available computer hardware and software. In general, it is a Linux PC cluster with hardware and software designed to optimize performance for a particular application. As a scientific user sees it, a PeTS system is literally a piece of lab equipment. It must:

1. Physically fit within a lab. We expect that each PeTS configuration will be fully contained (including I/O devices and/or interfaces to other instruments in the lab) within the space of a single standard 4-foot by 2-foot shelving unit and will not exceed the typical minimum ceiling height of 8 feet. Thus, a PeTS system will have a total volume of less than 64 cubic feet.
2. Use ordinary AC power and wiring. Typical US power is 110VAC with at least 15A per circuit. A PeTS system should use no more than two such circuits (ideally, just one circuit).
3. Be compatible with other uses of a lab or office. For example, it must operate at a comfortable room temperature and run reasonably quietly.
4. Have no "system administration" visible to the user. All regular system-maintenance tasks must be able to be performed remotely, and securely, via a wired or wireless link to the Internet.
5. Make the primary user-visible interface be the application, not an operating system. The user should only need to know the scientific problem to be solved and how to specify that problem using the application interface.
6. Yield performance comparable to or better than that available by remotely accessing a general-purpose shared supercomputer. While supercomputers that are faster for the

particular application may exist, the PeTS system is dedicated rather than shared, so it should be at least comparably fast for solving typical problems — and it will be far more responsive to the scientist.

From a computer-engineering point of view, PeTS systems are even more valuable. Portability and standard interfaces are a good thing at the user level, but can block the introduction of important new technologies at the system-design level. Because each PeTS system is a “black box” to its users and the turnkey design removes portability from being a concern, we have full freedom to employ radically new system hardware and software approaches. If a PeTS system delivers exceptionally good performance, it validates the new technologies that were employed in producing the system and makes them far more likely to be adopted by others.

2. What Are The Radical New PeTS Technologies?

We do not yet know what new technologies will prove to be most important, nor do we claim to know the full set of technologies that we will use in PeTS systems. Part of this proposal’s goal is to derive new technologies by examining PeTS applications. However, we already have developed a number of technologies that offer potentially huge benefits, but cannot be fully utilized without the systems-level design freedom a PeTS system provides.

2.1. New Interprocessor Communication Technology

One of the most important characteristics of any parallel machine is the way in which processing elements communicate. In this proposal, rather than simply using a conventional cluster network and software interface, we will use a wide variety of techniques, including multiple networks and pruned exhaustive and/or genetic searches of the design space, to optimize both the network hardware structures and the application software interface to that hardware. This level of application-driven customization of the interconnection network hardware and software represents a fundamental advance beyond the current state of the art.

2.1.1. Aggregate Function Networks

In a conventional cluster network, the network hardware consists of NICs (network interface cards), physical media (cables or fiber), and switches. The purpose of the network is to send data from one processing element to another, implementing point-to-point message (unicast) communication. Some networks also support broadcast or multicast communications, which allow a processing element to use a single network operation to send data to an entire group of processing elements. The primary performance criteria are point-to-point bandwidth and latency. These metrics are properly applied to the Internet. However, a parallel program is not equivalent to multiple independent sequential programs that sometimes interact, and thus different types of network operations are required.

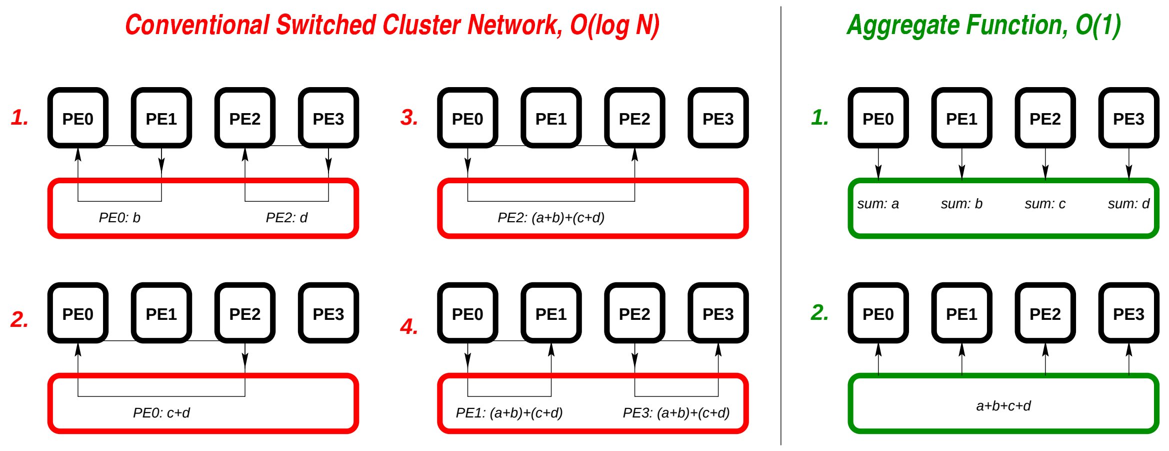
The distinguishing feature of a parallel program is that all parts of the program work to create a desired global state. Consequently, it is natural for the processing elements to operate on the global state of the computation. Throughout the history of parallel computing, efficient access to global state information has been viewed as an unachievable ideal. Most parallel codes have been carefully designed to minimize the need for such access, usually at the expense of significantly more complex program logic, redundant computations and data, and poorer load balance. Rather than suffer those costs, we have created a new communication model to make access to global state cheap: aggregate function communication.

The aggregate function communication model and its efficient hardware implementation are rooted in VLIW (Very Long Instruction Word) architecture and compiler code scheduling. In 1987, while trying to find a way to add a VLIW execution mode to the then-new PASM

(Partitionable SIMD MIMD) prototype supercomputer, we realized that the hardware that enabled PASM to implement SIMD was really a type of hardware barrier synchronization unit, and that such a barrier unit also could be used to support VLIW-style compile-time code scheduling (see **`http://garage.ecn.purdue.edu/~papers/PLCPC90/paper.html`**). Unfortunately, the fully generalized barrier mechanism seemed to lead to a fairly complex hardware implementation employing some form of associative memory for matching barriers. Although machines like the Thinking Machines CM5 and Cray T3D implemented a subset of the barrier synchronization mechanism that we described, a fully general high-performance barrier mechanism was not implemented until much later.

In late 1993 we discovered a way to build the fully general barrier mechanism using multiple AND trees and an unusual, but very simple, communication mechanism designed to allow processors to rapidly agree on which processors should participate in each barrier. In February 1994, we built the first Linux PC cluster for the purpose of testing this new barrier hardware: PAPERS, Purdue's Adapter for Parallel Execution and Rapid Synchronization. Although we also had Ethernet connections between the PCs, Ethernet performance was very poor, so we tried using the barrier mask communication hardware for general communication. It worked stunningly well. Since that time, through 18 generations of PAPERS hardware, we have generalized and refined this new communication model in which each processor gives the network a datum and an opcode, the network computes the desired function on data aggregated from an entire group of processors, and the computed result is returned to the requesting processor(s).

Aggregate functions directly implement a variety of familiar operations, such as barrier synchronization, SIMD any and all, broadcast, reductions, and scans (parallel prefix). For example, consider performing an add reduction (summation) on a conventional network versus using an aggregate function network:



Beyond these familiar operations, aggregate functions include many new operations on global state. For example:

- **`Putget`** is an aggregate operation, implemented using multiplexors, in which each processor outputs a datum and specifies from which processor it would like to read the datum; this "backward" routing means that a single **`putget`** operation can implement any permutation (or permutation with replication) without any contention. Personalized all-to-all can be implemented optimally as a series of **`putget`** operations.
- A new class of aggregate operations allows processors to vote for the resources they wish to use. Perfect global data summarize the relevant resource requests, so operations on these resources can be statically scheduled at runtime, thereby achieving contention-free peak

performance for use of these resources. Thus, voting operations can dramatically improve access speed for shared resources ranging from data structures to message-passing network ports.

- Aggregate signals allow an individual processor to cause a group of processors to perform a specified action. Generalizing the Cray T3D's "Eureka" operation, these signals can be used to inform a group of processors of any type of unanticipated change in the global state of the program.

This model has been used by dozens of institutions worldwide, but only in very limited ways. Although we have implemented an MPI 2.0 that uses aggregate function communications, MPI includes only a few aggregate functions and conventional wisdom is that they are expensive... so applications rarely use them. In fact, our simple public-domain hardware makes many aggregate-function operations cheaper than sending a single minimum-length point-to-point message on a high-performance message-passing network, such as Myricom Myrinet, Giganet CLAN, or Dolphin SCI.

Aggregate function communication has not found more widespread use because it is a radically new model. **Most application codes must be entirely restructured so that, instead of using convoluted techniques to make do without global state, they take full advantage of cheap, perfect, global state information.** We believe that rewriting various important applications will not only deliver better performance for these applications, but also will establish the aggregate function model as a key component of the design of future parallel algorithms.

We expect each PeTS system to incorporate at least two networks: a fairly conventional point-to-point network for sending blocks of data from one PE to another and an aggregate function network for N-way access to global state.

2.1.2. Interprocessor Hardware/Software Interface

Another key issue in obtaining good communication performance is the minimization of software overhead in performing network operations.

Library interfaces such as PVM and MPI generally use multiple layers of software to provide their relatively high-level abstractions. It can be argued that these abstractions are worth the overhead because they greatly simplify the task of coding complex parallel algorithms, but the overhead is substantial. If you would not accept multi-layer software for talking to a floating point coprocessor within your system, why are you willing to accept these layers when talking to another processor that is working on the same program?

For example, using most versions of PAPERS, a complete barrier synchronization takes about 3 microseconds. That time includes all hardware and software overhead. An operating-system call typically takes an order of magnitude longer than that; even a subroutine call introduces over a microsecond of delay unless the subroutine is fully contained in cache. *We expect to use single-layer direct I/O accesses for nearly all interprocessor communications.*

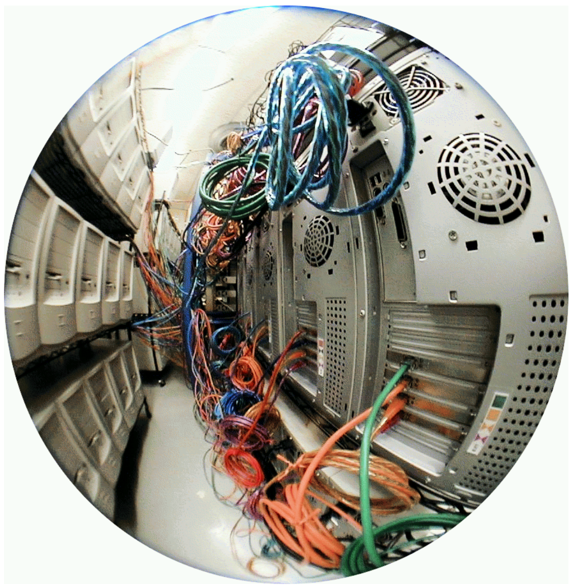
2.1.3. Global Communication Optimization

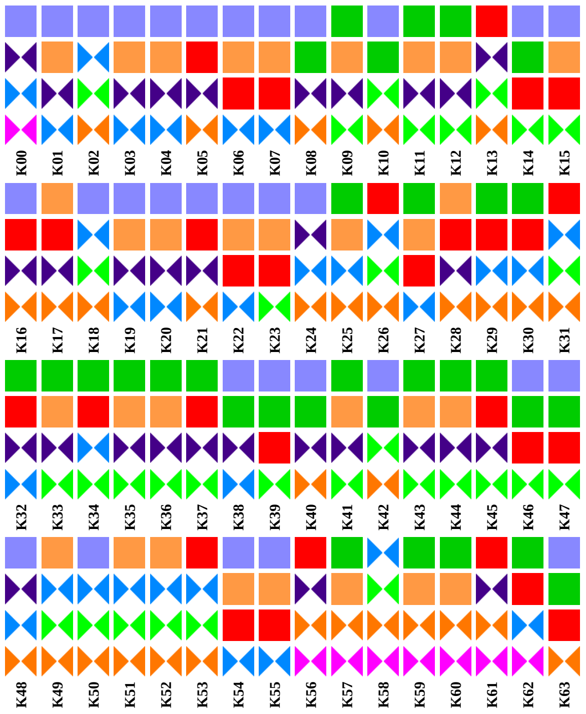
As suggested in our discussion of aggregate function networks, it is possible to use low-latency, low-bandwidth, aggregate function communications to obtain global information with which to schedule accesses to be made through a high-latency, high-bandwidth, conventional switched network. Beyond that, an exciting possibility is that even the optimal basic structure of the high-bandwidth network can be derived directly from the communication behavior of an application.

Our first experiments with manipulating conventional network properties to meet a set of application-specific optimization criteria were in 1994. The global router network in MasPar's MP1 and MP2 supercomputers used an unspecified hardware algorithm to resolve routing conflicts among the 16,384 processing elements. Although the hardware generally worked well, tests showed that the hardware router scheduling required 70 router cycles to implement bit-reversal permutation communication. Using a genetic search program, we were able to explicitly regroup the router cycles so that the same operation could complete in just 23 router cycles.

Just in the past month, we have developed a similar technology for designing a new class of cluster networks, called "flat neighborhood networks," that offer single-switch latency for small messages while providing virtually the same bisection bandwidth as a full crossbar, using multiple NICs per PC and switches that are only a fraction of the width of the full cluster. Again, a genetic search program optimizes the network to maximize bandwidth for specific communication patterns.

Our newest 64-PC cluster, KLAT2 (Kentucky Linux Athlon Testbed 2), is the first machine ever built with a flat neighborhood network. The 64 machines each contain four 100Mb/s NICs that are wired to nine 31-way color-coded switches (subnets) in the following pattern.




K00 K01 K02 K03 K04 K05 K06 K07 K08 K09 K10 K11 K12 K13 K14 K15
K16 K17 K18 K19 K20 K21 K22 K23 K24 K25 K26 K27 K28 K29 K30 K31
K32 K33 K34 K35 K36 K37 K38 K39 K40 K41 K42 K43 K44 K45 K46 K47
K48 K49 K50 K51 K52 K53 K54 K55 K56 K57 K58 K59 K60 K61 K62 K63

KLAT2's flat neighborhood network
Above: physical wiring
Left: neighborhood pattern

No switch is connected to another switch (except in that all nine switches are connected to a tenth switch used only for multicast and hosting two "hot spare" PCs). This random-looking wiring pattern was genetically designed so that (1) every pair of PCs shares at least one switch and (2) bandwidth is maximized for PC pairs that share the same row or column in an 8x8 arrangement. The result is over 25Gb/s bisection bandwidth for less than $8,000 spent on network hardware, whereas $250,000 spent on Giganet CLAN hardware would have yielded only 9Gb/s bisection bandwidth.

To take full advantage of the optimized network, our genetic network design programs also create customized routing information for each PC pair. In comparison, the channel bonding developed for Linux under the original Beowulf effort requires that all NICs within each PC be connected by separate, but identical, topology. Our technique does not require that NICs go to separate switches, nor does it require that the topology be identical (or even similar). Further, although channel bonding is essentially invisible to the user, it is done at a very low level by duplicating the same MAC address for all the NICs in each PC; the result is that application code cannot schedule the use of specific NICs. Even if we have switches that are the full width of the cluster, this scheduling ability can make a huge difference in the performance of the network.

We expect each PeTS system to have a network that is optimized, both in structure and in scheduling of accesses, to meet the communication needs of the target application.

2.2. New Compiler Technology For SWAR

Over the past five years, nearly every microprocessor design has added instruction-set extensions intended to provide vector-like parallel processing without requiring major modifications to the processor architecture. First, there were a variety of integer extensions aimed primarily at software MPEG decoding: Hewlett-Packard's PA-RISC MAX; Digital Equipment Corporation's Alpha MAX (now called MVP); Intel Corporation, Advanced Micro Devices (AMD), and Cryrix Corporation's x86 MMX; Silicon Graphics MIPS MDMX; and Sun Microsystem's SPARC V9 VIS. With the increasing popularity of 3D games like *Doom* and *Quake*, vector floating-point support was added to PC processors: AMD's 3DNow!, Intel's KNI (now called SSE), and Motorola's PowerPC AltaVec (also known as the Apple PowerPC G4 Velocity Engine). Generically, we call all these extensions SWAR (SIMD Within A Register).

The basic concept of SWAR is that operations on relatively wide k-bit registers can be used to speed up computations by performing SIMD parallel operations on n field values, each k/n-bits long. New instructions partition data paths and function units for specific operations. For example, MMX supports operations on 8, 16, and 32-bit fields within 64-bit registers; 3DNow! also manages two 32-bit floating-point fields within the same 64-bit data paths. Thus, in processors like AMD's K6-2 or Athlon, using two pipelines, MMX and 3DNow! offer 16x improvement in integer speed and 4x improvement in floating-point speed. However, because SWAR extensions are very machine dependent, have many restrictions, and are primarily focussed on hand-coding a few specific applications, making good use of SWAR technology can be very awkward.

Since 1996, with help from both Intel and AMD, we have been developing compiler technology that can effectively manage the complexities of generating good code for SWAR targets. The result, publically available since 1998, is a compiler for a SIMD-parallel extension of C called SWARC (see **`http://shay.ecn.purdue.edu/~swar/`**). The Scc compiler uses a variety of new compiler techniques, including simultaneous optimization of register allocation, code scheduling, and addressing-mode selection. Local optimization uses a detailed, nearly cycle-accurate, model of the target processor pipelines to drive a pruned exhaustive search (that can be truncated after a user-specified period of time). This extremely aggressive compiler technology is needed primarily because SWAR places even greater demands on the processor/memory interface, which often is the limiting factor on processor performance even without using SWAR. Botching even a single SWAR register assignment can yield slowdown rather than speedup.

Even with the Scc compiler, SWAR is not trivial to use. Relevant portions of the application must be rewritten in a SIMD style, with relatively short vectors. Further, SWAR floating-point support gives 32-bit single-precision arithmetic a substantial performance edge over the usual 64-bit (or, for x86, 80-bit) double-precision arithmetic. Thus, it becomes necessary to carefully analyze the precision required throughout an application, perhaps replacing some algorithms with others that can better utilize lower-precision operations.

We expect to re-tool at least some portions of each PeTS system's application code to take advantage of the SWARC compiler and SWAR technology.

2.3. New Attached-Processor Technology

Although people often focus on PCs as *the* compute engines within a cluster, a wide range of attached processors now can be purchased as relatively cheap PCI-bus cards. Some of these cards offer remarkably good performance for specific tasks, but typically lack the generality of the PC's main processor. Examples include:

- Audio and video cards with custom processors to accelerate typical tasks. For example, some high-end video cards are capable of surprisingly fast single-precision floating-point matrix operations... because these operations are needed to accelerate the same games that inspired the development of 3DNow!, SSE, and AltaVec.
- Cards with multiple DSP (Digital Signal Processor) chips. Due to the difficulty of upgrades and the relatively poor performance of these processors compared to PC processors like the Athlon, these cards are not as much of a bargain as they once were.
- Specialized attached processors, such as programmable logic devices and content addressable memory (CAM).

Of these, specialized attached processors offer the largest potential performance boost, so our efforts will focus on these. In particular, working with Aeroflex (**`http://www.aeroflex.com/`**), we have started to develop compiler technology and Linux interface software for using the UTMC eCard "Distributed Query Processor" CAM hardware.

On a single PCI card, with an estimated retail cost under $1,000, the eCard contains a 500 MFLOPS MIPS processor and two CAM engines. Each of these CAM engines can perform an associative access in as little as 100 nanoseconds. Search keys can be from 1 to 32 bytes in length, and a variety of match types are supported: exact, prefix, and proximity (by Manhattan or Euclidean distance).

Although the value of these cards as database accelerators is obvious, we also will be developing compiler support for using them as applicative caches. Applicative caching, also known as "memoizing," can be applied to any function that has the property that the return value is always the same for the same set of function arguments. Each time the function is called, the argument tuple is looked up to see if the value has been computed before; if so, it is returned from a software-managed cache rather than recomputed. Although the cost of looking up argument tuples often outweighs the benefit on conventional processors, these CAM units dramatically reduce the cost of argument-tuple lookup. There is still a relatively high cost in communicating between the host PC processor and the eCard, but we can minimize the impact of this by vectorizing the argument tuples and pipelining the CAM execution. Our compiler technology will perform this transformation on any functions marked by the programmer.

The result is that applicative caching might be productively used for functions as simple as arctangent. Further, simple interpolation techniques can be used to allow similar speedups for imprecise lookups. Thus, we might be able to use CAM to significantly accelerate a far wider range of computations than one might initially suspect.

We expect that each target application will be scrutinized to see whether specialized attached processors can improve computational efficiency; where such processors are beneficial, we will provide compiler and software interface technology to support their use.

3. The Research Focus

Although building PC clusters has become very popular, the sad truth is that very few PC clusters take advantage of any of the configuration options and technologies that could improve the performance beyond that of a traditional supercomputer. A typical "Beowulf," trivially assembled as PCs + high-bandwidth network + MPI, is effective only on a relatively small range of applications and, even for those, primarily yields good performance per unit price rather than the best performance.

Even if a scientist has a problem that can run acceptably well on a conventional PC cluster, it is very difficult for scientists to configure, use, and maintain such a system. They need to be very computer literate and to spend a great deal of time working on computing rather than working on their scientific research. Every aspect of a PeTS system must be designed to allow a scientist to focus on science rather than on computing.

Thus, we must solve three major research problems:

1. How does each application work and how can it be radically restructured? We need to know this to determine which new cluster technologies can be applied.
2. How should we configure a system using low-cost COTS and/or freely-available components to get better-than-conventional-supercomputer performance for the application? Here, we will further develop and showcase our new cluster systems technologies.
3. How can we make the resulting system truly be a turnkey solution, an apparently simple piece of "laboratory equipment?" This includes simplification of the cluster construction process so that building and installing a highly-tuned PeTS system will be no more complex for the scientist than installing any other piece of laboratory equipment. It also includes long-term system administration issues.

3.1. Working With The Applications

The University of Kentucky has a well-established **Center for Computational Sciences** (**`http://www.ccs.uky.edu/`**). This center, directed by Co-PI John Connolly, will be used to identify scientific applications that are most appropriate for us to target with PeTS systems and which scientists our group should work with in developing these systems. Every six months, when a new list of the 500 fastest supercomputers in the world (**`http://www.top500.org/`**) is created, only about ten academic sites have supercomputers powerful enough to rank; the University of Kentucky's center has been listed in five of the last eight lists. By its charter, this center provides free supercomputing facilities and consulting support to the entire university (including the Chandler Medical Center) and to NCSA Alliance and NSF EPSCoR users elsewhere. Connolly is also the director of the Kentucky Experimental Program to Stimulate Competitive Research (EPSCoR). In addition, the PI also has ongoing research relationships with several important applications research groups at Purdue University. Thus, we have access to a very wide application base, excellent long-term relationships with the scientists developing and/or using these codes, and an administrative structure that can help manage the placement and user testing of PeTS systems.

We expect each PeTS system to take about one year to create, but we will pipeline the process to complete one or two PeTS designs each year. In addition to sharing a substantial pool of systems-level development resources dedicated to this proposal, each PeTS design team will include a post-doc, a graduate RA, and an undergraduate RA who will work directly with the scientist or engineer selected to guide conversion of that application into a PeTS system. A scientist who agrees to work with us not only receives a PeTS system about a year later, but also gets the undivided attention of three team members. Further, these three students receive training that makes them experts in that application, greatly improving their future job prospects.

The proposed research not only will produce PeTS systems for a variety of scientific problems at the University of Kentucky, its NCSA Alliance and NSF EPSCoR partners, and other institutions, but also will make it trivial for others to duplicate those PeTS systems in their laboratories. The budget is designed to make this widespread impact possible: it includes a secretary to serve as a single point of contact for all the PeTS project participants and users, money for travel to conferences and to make site visits to partners, and money to cover costs of publishing and distributing the results of our work (such as journal page charges and replication costs for PeTS software CDs and manuals).

3.1.1. Selection Of Applications

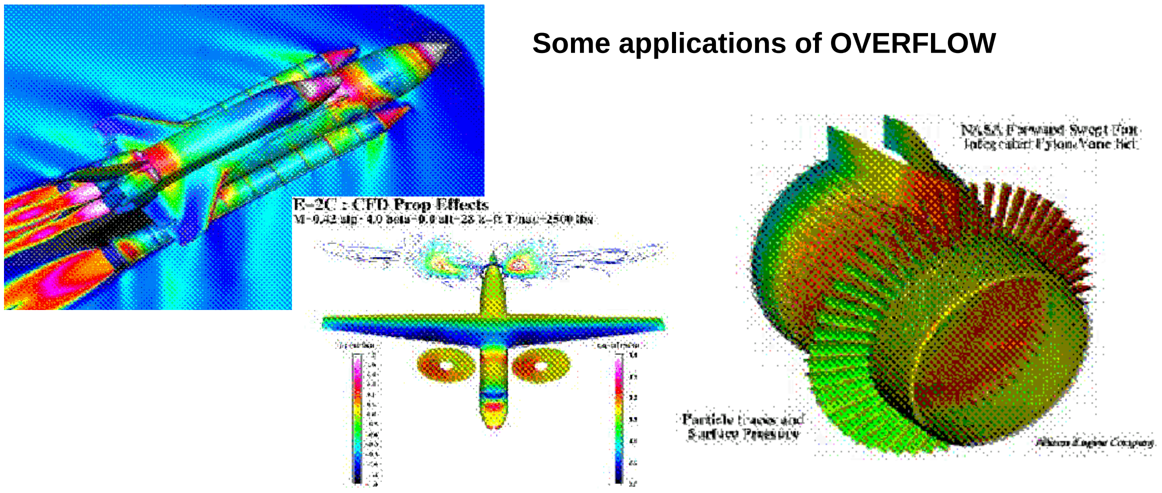
We already have identified a number of promising applications, ranging from the molecular modeling of proteins being done by Carol Post at Purdue University (see **`http://www.pharmacy.purdue.edu/mcmp/post/post.html`**) and Trevor Craemer at the University of Kentucky (see **`http://www.uky.edu/Medicine/Biochemistry/department/faculty/creamer.html`**) to the computational electromagnetics being done by Stephen Gedney at the University of Kentucky (see **`http://www.engr.uky.edu/~gedney/`**). However, PeTS target applications will be selected by a process of open invitation, followed by evaluation by the principals of this proposal. Each potential PeTS application will be evaluated on the basis of:

- Potential of the application to be qualitatively improved through the use of our new technologies.
- Ability of the application submitter to work with us in radically restructuring the code. We want to work with scientists and engineers who not only use their application code, but also have taken an active role in the development of the code.
- Appropriateness of the PeTS concept for the application. Many scientific and engineering applications simply cannot use turnkey solutions because modifying the code is part of the normal course of that research. We are looking for applications that, once implemented as PeTS systems, can be widely used with little or no change to the hardware and software of the PeTS system.

We anticipate accepting either one or two new PeTS target applications each year. Rather than specifying a fixed starting date for new projects, we will pipeline the development process, accepting a new PeTS target application as soon as the previous PeTS target has progressed enough to allow us to move resources to a new target.

3.1.2. The First PeTS: The OVERSET Tools For CFD Analysis

So that PeTS development can begin immediately upon receipt of funding, and to ensure that the first PeTS system will clearly demonstrate all the basic concepts and benefits of PeTS technology, we have identified the OVERSET tools for CFD (Computational Fluid Dynamics) analysis as the first PeTS target application. This code has many applications, a few of which are shown here:


E-2C : CFD Prop Effects
NASA Forward Swept Fan Integrated Pylon/Vane Set
Particle traces and Surface Pressure
Allison Engine Company

Some applications of OVERFLOW

Despite requiring extensive computing resources, CFD analysis has become a standard tool for a wide range of engineering design tasks. For example, a complex vehicle flying at supersonic speeds generally requires CFD analysis over several million grid points to resolve shocks and other flow structures correctly. Such a computation requires enormous computer resources, beyond what any single scalar or vector computer can provide.

National supercomputer centers have facilities that can support analysis of large-scale CFD problems, but access to these facilities is limited, and organizations other than government laboratories and research universities might not have any access at all. Shared-memory supercomputers, such as the SGI Origin 2000 and HP-Convex SPP-2000, have become the dominant choices for US industry to use in analysis of somewhat less complex CFD problems. Unfortunately, the cost of the these computers is still prohibitively high for most universities and many companies; only large corporations, such as those in the aerospace, automobile, and chemical industries, can afford to make use of these shared-memory parallel computers for their CFD designs. Smaller companies, and researchers who wish to work on local machines, have been struggling to perform their CFD analysis using high-performance workstations that are slow, but still not cheap.

The obvious solution is to port CFD codes to run on inexpensive PC clusters — “Beowulfs.” Indeed, there are now highly portable CFD design codes that use PVM or MPI for communication. Dr. Dennis Jespersen of NASA-Ames recently ported the OVERFLOW code to workstation clusters using PVM or MPI. However, the use of either of these libraries implies significant unnecessary overhead. For example, CFD interprocessor communication tends to follow specific patterns; using this information, we can optimize the design of the network hardware and use software to generate globally optimal schedules for network transmissions. There also are numerous opportunities to take advantage of SWAR instructions. With significant code restructuring, it may even be possible to use content-addressable memory (CAM) attached processors to speed-up the CFD code by caching partial results. In summary, portable CFD codes run slower on conventional PC clusters (even those using expensive network hardware) than they do on traditional supercomputers, but there are a number of aggressive systems-level optimizations that we can apply to make a cheap PeTS system outperform a traditional supercomputer on CFD analysis.

It is not sufficient that the application be important and amenable to our techniques; we also need an application user/developer who can work with us. That person is George Huang, whom we have listed as a Co-PI to make clear his commitment to this project.

Huang has been a major contributor in the development of NASA's highly-acclaimed *OVERSET tools for CFD analysis.* In 1998, his work on this package was recognized by making him co-winner of the "Software of the Year" Honorable Mention award (see **`http://www.hq.nasa.gov/office/codei/swy98win.html`**). Development of the OVERSET tools is currently supported by NASA and CHSSI (the DoD's Common High Performance Computing Software Support Initiative, see **`http://rotorcraft.arc.nasa.gov/cfd/CFD4/New_Page/Home.htm`**). However, the PeTS effort will be the first integrated attempt to restructure both the application code and system hardware and software to optimize the performance of this application.

The OVERSET Tools are a collection of overset grid CFD software programs developed at NASA-Ames (under Dr. Pieter Buning of NASA-Langley), including Collar Grid Tools (grid manipulation and surface grid generation), HYPERGEN (volume grid generations), PEGSUS (overset grid hole cutting and boundary interpolations), DCF3D (an alternative to PEGSUS, especially for moving grids), and FOMOCO (force and moment calculation for overset grids). The solver of overset tools is OVERFLOW, which has been optimized to run efficiently on the Cray C90; production versions also exist for the IBM SP2 and workstation clusters. Primary customers for the OVERFLOW flow solver and related software include major aerospace companies, various DoD research centers, and NASA focused programs for Advanced Subsonic Technology (AST) and High Speed Research (HSR).

OVERFLOW is a fairly large code, combining 120,000 lines of Fortran and 1,500 lines of C. There are 1,200 subroutines and 1,100 files. It was written in a vector-oriented style, with the preferred architecture for execution of the program being a vector supercomputer. A Cray C90 vector processor executes the code extremely well — over 400 MFLOPS. The IBM SP2, Cray T3E, and SGI Origin 2000, achieves only 20 to 90 MFLOPS per processor. Our preliminary results using a single Athlon PC were nearly twice as good, with the potential to surpass the C90 per-processor performance if 3DNow! can be used. Further, our radical new approach to network design should allow that performance to scale very well using more PCs. It is clear that a well-designed PeTS implementation could make of state-of-the-art CFD analysis significantly more accessible.

Finally, because the OVERSET CFD Tools are widely available and run on several different platforms, we have an excellent basis for comparison with PeTS performance.

3.2. PeTS Configuration: Widening And Exploring The Design Space

The problem of configuring a computer to efficiently execute a particular application is conceptually easier than that of creating a fast general-purpose computer, but the design search space must be significantly larger if low cost and high performance are to be achieved for applications that are not inherently well matched to conventional "Beowulf-class" clusters. An extremely detailed understanding of the application also is necessary, since many of the most important performance improvements will require at least minor algorithm changes in response to architectural changes in the computing system (such as restructuring a matrix multiply or substituting a genetic search for simulated annealing).

3.2.1. Widening The Design Space

Hank Dietz, the PI, has a long history of modifying and/or creating system software and hardware to expand the range of problems that computing systems can execute efficiently. In February 1994, his group built the first parallel processing Linux PC cluster — 8 months before "Beowulf" — and this cluster augmented the purely COTS PCs and Ethernet hardware with a secondary network that his group designed and released. He also has played a major role in popularizing Linux PC-based parallel processing through his authoring of the Linux Documentation Project (LDP) **Parallel Processing HOWTO**, which covers all aspects of parallel process using Linux PCs, including shared memory, cluster, SWAR (SIMD Within A Register), and attached-processor systems.

Dietz's group, based at Purdue University until he moved to a chaired position at the University of Kentucky in Fall 1999, has produced many widely used research products. Fully *public-domain* complete hardware designs and source-code releases from his group include:

- The **Purdue's Adapter for Parallel Execution and Rapid Synchronization (PAPERS) aggregate-function network hardware**, which implements a variety of operations on global state (barrier synchronization, reductions, scheduling/load balancing, etc.) across a Linux PC cluster with latency as low as a few microseconds and less than $100/PC additional cost:
  `http://garage.ecn.purdue.edu/~papers/Hardware/`
- The **Aggregate Function Application Program Interface (AFAPI)**, a highly-portable aggregate-function communication software interface and optimized implementations for target systems ranging from PAPERS clusters to generic SMP UNIX systems:
  `http://garage.ecn.purdue.edu/~papers/AFAPI/`
- **VWLib**, the Video Wall Library designed for driving computational video walls (e.g., for scalable scientific visualization) using Linux PC clusters with aggregate function networks.
  `http://aggregate.org/`
- The **Purdue Compiler-Construction Tool Set (PCCTS, Antlr, etc.)**, which greatly simplifies creation of specialized optimizing compilers and other user interfaces:
  `comp.compilers.tools.pccts`
- The **SIMD Within A Register programming model (SWAR), C language dialect (SWARC), and optimizing compiler (SCC)**, which allow scientific applications to take advantage of MMX and 3DNow! to achieve up to 16X integer speed and over 4X single-precision floating point speed on unmodified PC hardware:
  `http://shay.ecn.purdue.edu/~swar/`

In this proposal, Dietz, and Co-PIs Raphael Finkel and Jim Lumpp, one graduate RA, and one undergraduate RA, will continue to significantly broaden the range of application parallelism that can be effectively used by improving and/or creating system software (mostly compilers and drivers) and hardware. This effort will focus primarily on four of the weakest points in current cluster performance:

1. Compiler technology for better utilizing *new processor features* (such as the SWAR support and future 64-bit extensions)
2. Hardware and software to *improve communication performance* for the key operations found in the applications
3. Hardware and software to utilize low-cost *attached processors* (DSP coprocessors, Content-Addressable Memories, FPGA-based circuits)
4. Application-specific algorithm design for PC clusters

The PI's previous PC-cluster development work has been supported by a number of key industrial partners, including major contributions from Intel, AMD, and Microsoft. In this proposal, in addition to conducting our own development work, we expect to be directly working with at least two industrial partners: AMD and Aeroflex.

AMD (Advanced Micro Devices `http://www.amd.com/`) has incorporated a number of mechanisms in their processor designs that can significantly improve both cost and performance of PC hardware for use in PeTS. The most obvious of these features is the support for various types of SWAR parallelism within the processor chip. In April 1997, about a month before Intel released the Pentium II, AMD released the K6 with **MMX (MultiMedia eXtensions)** integer SWAR instructions. Although AMD's processors had poorer scalar floating-point performance than Intel's, the AMD K6-2, released in mid-1998, added **3DNow!** floating-point SWAR support that gave up to 4 times the performance of Intel's still-more-expensive processors. Intel's Pentium III SSE (Streaming SIMD Extensions) not only came out half a year later than AMD's 3DNow!, but also suffers from a variety of secondary problems that, in our tests, make performance harder to get from SSE on a Pentium III than from 3DNow! on a K6-2. AMD's Athlon improves a number of secondary features, fills a few gaps in the MMX and 3DNow! instruction set, and upgrades the double-precision floating-point support to easily surpass Intel's best offerings. AMD plans for their future processors to add an enhanced **"Technical Floating Point"** mechanism and full **64-bit support**. In summary, to gain an edge against Intel, AMD has been pursuing features that give their processors a significant edge in scientific computing applications — and AMD has directly supported our work toward making it easier for the new features to be used in that way. For example, AMD donated a cluster of Athlons to us in Fall 1999 and also supplied us with the Athlons for KLAT2 (assembled April 11-15, 2000).

As discussed earlier in this proposal, **Aeroflex (`http://www.aeroflex.com/`)** will be helping us to apply a new Content Addressable Memory (CAM) system as an attached parallel processing engine for each node in a PC cluster. These CAMs, which will be available as inexpensive COTS devices, can apply a variety of fuzzy matching mechanisms to large (up to 2GByte) data sets. The unique capabilities of this CAM might make it possible to speed up not only search-oriented applications, but also more general computations that can use applicative caching ("memoizing") or table lookup and interpolation. To facilitate our development of both a low-latency Linux device interface and application-level CAM software, Aeroflex will build and donate an appropriate number of PCI-interface CAM systems and will provide us with technical help including detailed technical specifications, a CAM simulator, and access to their engineering staff. They also plan to make the PCI CAM cards available as under-$1,000 COTS units so that others will be be able to use the CAM software that we develop.

3.2.2. Exploring The Design Space

While the above section discusses how we will develop a wide range of additional abilities for use in PC-based computing in general and PeTS in particular, the problem of selecting which features to include in each PeTS system must also be solved. Picking the best configuration for each application also requires extensive experimental evaluation of possible PeTS system designs.

Dietz and Lumpp will oversee the process of characterizing the range of cluster architectures and finding the best match for each application. There will be one graduate RA and one undergrad RAs dedicated to this process; these students will work with the application teams to create an optimized pairing of application coding and PeTS system configuration.

Configuration variables include PC processor type, use of instruction-set extensions, use of attached processors, memory size, disk size and quantity, type(s) of interconnection between processors, and I/O devices. In contrast to earlier efforts of various groups to benchmark different motherboards, compatible NICs, etc., our focus is on *higher-level* (often architectural) differences that have significant performance implications that last well beyond the next board revision. We also will design and build simple custom hardware if an application has a performance issue that cannot be solved in any other way — as we did in developing PAPERS.

To experimentally evaluate the possible PeTS system designs for each application, we need to have a **development cluster** that is capable of emulating any of the possible designs, which essentially means that it will contain the union of all viable components and must be easily reconfigurable to selectively disable the components that will not appear in a particular emulated system. Although our industrial partners are expected to donate some of the most expensive computer hardware components, it will be necessary to purchase other portions of the hardware ranging from traditional PC components like power supplies, cases, and disk drives to shelving units and mounting hardware. We have budgeted $100K/year for components of the development system and one half-time technical staff member for its construction and maintenance.

As portions of the development system hardware become impossible to duplicate with new purchases (i.e., become obsolete), where appropriate, those *portions will be "recycled" by making them into more PeTS systems*. These additional PeTS systems will use configurations developed earlier in the project and will be given to additional scientists at the University of Kentucky or its NCSA Alliance and NSF EPSCoR partners. Thus, very few resources are wasted in frequently upgrading the development cluster.

3.3. Making A PeTS Configuration Into A Turnkey System

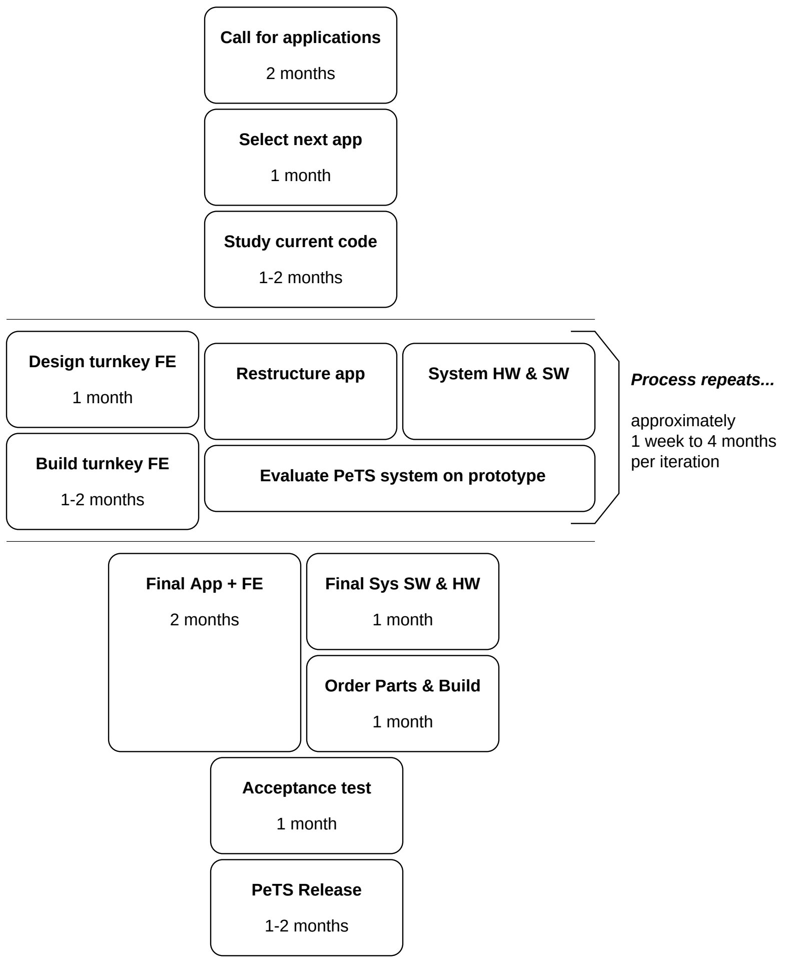
Although it is our final step in the development of any PeTS system design, the most important characteristic of a PeTS system for its scientific user is probably how easy it is to use:

- The user should interact only with an application interface that expresses problems and answers in the scientist's idiom
- There should be no regular system administration visible to the user; system administration should be minimal and should be able to be conducted remotely via the Internet (e.g., by the staff of this project)
- The system set-up and installation should be as effortless as possible — installed by a half-time technical staff member for the first system, but also documented and distributed with "cookbook" simplicity for others to build clones of the first system

Co-PI Raphael Finkel, a systems researcher with an outstanding record in the automation of system administration for groups of machines (e.g., clusters), will lead the effort in making the application all that the scientist needs to think about. Toward this, he will supervise one graduate RA and one undergraduate RA, as well as working with the application teams to integrate their application interface into the system. Thus, Finkel and his students will be responsible for operating-system issues, including both the automation of system administration and the general packaging of the software as a turnkey environment and PeTS software distributions.

4. Development Timeline

All the PeTS systems to be developed under this proposal will share much of the systems-level research, which will be progressing continuously throughout the five-year period of the work. We expect that the aggressive use of new technologies and radical restructuring of applications in producing PeTS systems will result in dramatic variations in the time to produce each PeTS design. However, it is useful to examine the process that a typical PeTS design will go through:



From call for applications to release of the final system, we expect a typical PeTS system to require a development period of just over one year. Typically, there would be between two and four PeTS systems under development simultaneously, each at a different stage in the development pipeline. We have “primed the pipe” by working-out the first few stages of the CFD PeTS system beforehand, so the delay in reaching our desired steady state should be no more than six months.

The pipeline stages are fairly straightforward. Whenever we see a gap in the pipeline that would allow us to start a new project, we will issue an open invitation for scientists and engineers to propose applications for PeTS implementation. After selecting the most appropriate application, we will begin a detailed analysis of the current code versions. From that point, development of a turnkey front end can begin. Likewise, we can begin the iterative process of restructuring the application and system software and hardware, evaluating the resulting potential PeTS design using our development cluster, and then revising the design to improve performance. We expect this iterative process to continue for about 4 months on average, but expect substantial variation. The next steps finalize the design and construct the actual PeTS system, which is then put through acceptance tests that compare it to other implementations of the same application. Finally, the complete application and system design will be released as a finished product, easily replicated by other potential users. It is in this last stage that we also will publish on the techniques used.