WebFlow: A Visual Problem Solving Environment for Wide-Area, Heterogeneous, Distributed High Performance Computing.

Geoffrey C. Fox, Wojtek Furmanski, Tom Haupt, Scott Klasky

Northeast Parallel Architectures Center, Syracuse University, Syracuse, NY
{gcf,furm,haupt, scott}@npac.syr.edu, http://www.npac.syr.edu

Marina Chen

Department of Computer Science, Boston University, Boston, MA
mcchen@cs.bu.edu http://cs-www.bu.edu

James Cowie

Cooperating Systems Corporation, Chestnut Hill, MA
cowie@cooperate.com http://www.cooperate.com

James Browne, Manish Parashar

Department of Computer Science, University of Texas, Austin, TX.
{browne, parashar}@cs.utexas.edu

Proposal to the National Science Foundation
New Technologies Program
July 2, 1996

PROJECT SUMMARY

We propose two cooperating software layers, WebVM and WebFlow, that will use familiar, broad-based Web technology to support high-level design and management of distributed High Performance Computing applications. The Web offers what HPCC has so far lacked: universal availability of standard client and server software, protocols and familiar visual user interfaces.

Adapting Web tools, protocols, and interfaces does not equate to chasing a computational fad. We believe that the explosive turnover in Web languages and protocols has begun to cool; HTML, Java, and VRML have now emerged as a relatively stable basis for long-term planning. Indeed, we argue that some basic lessons learned from the Web must soon be incorporated into HPC software, if we are to build, maintain, and use complex HPC systems effectively to serve national and defense needs.

Furthermore, our use of the term "Web" does not imply that systems which exploit Web technologies should be built atop the internet. In fact, use of Web technology for particular application domains and particular network systems has achieved great prominence recently. HPCC research "Intranets" consist of high-bandwidth, low-latency links engineered to maximize reliability and throughput. Our primary challenge is to retarget the Web-computing model (multi-server text, video, and executable software content) to meet the performance and reliability requirements of HPCC hardware and applications.

The HPC challenge is not one that is expected to be taken up by commercial Web software vendors in the foreseeable future. Too many significant research issues must be solved before industry will invest in high-performance versions of Web software — an unorthodox approach to a relatively small market.

The Web software tools this project builds will be critical to the evolution of HPCC precisely because they offer some solutions to these issues which are critical but not part of the commerical mainstream: how to

– manage the complexity of constructing distributed high performance applications software,

– enable distributed collaboration among HPC experts, and

– promote reuse and leveraged development of portable HPC software.

The Web Virtual Machine (WebVM) defines a mesh of interacting Web servers, cooperating to match published hardware and software resources with browser clients in search of cycles and expertise. WebFlow adds a dataflow programming paradigm atop WebVM, complete with tools for visual authoring of computational dataflow graphs and integration of software modules coded in traditional HPC languages. WebFlow addresses coarse grain software integration and dynamic/adaptive use of distributed resources, leaving traditional HPCC technologies such as MPI and HPF for the local, latency sensitive parts of the computation. To demonstrate our approach, we will stress-test the WebVM/WebFlow programming environment with a challenging application: a PDE problem solving environment using adaptive mesh refinement.

We initiated this process in 1995 with early pilot projects such as WebTools for personal CGI-extended Web servers and Webwork for linking computational servers to form distributed processing surfaces. Our first Webwork application, RSA130 Factoring-by-Web, was awarded the most geographically dispersed and heterogeneous solution in the 3rd Annual High Performance Computing Challenge Award contest at the Supercomputing '95 conference. This experiment, along with the growing interest in Web based HPCC within the Grand Challenge Communities have converged in our proposed wide-area virtual machine architecture (WebVM) and HPC coarse-grained visual programming paradigm (WebFlow).

The proposed systems will simplify the task of designing, maintaining, and reusing HPC applications software within HPC Intranets. In particular, we suggest that these technologies will be very valuable in the sort of geographically distributed data intensive applications and collaborations envisaged in the recompetition of the NSF supercomputer centers.

TABLE OF CONTENTS

	Section	No. of Pages
A	Project Summary	1
B	Table of Contents	1
C	Project Description	14
D	Bibliography	3
E	Biographical Sketches	13
F	Summary Proposal Budget	13
G	Current and Pending Support	9
H	Facilities, Equipment and Other Resources	1
I	Special Information/Supplementary Documentation	0
J	Appendix	0

PROJECT DESCRIPTION

1 Introduction, Motivation and Overview C-2
1.1 WebVM and WebFlow C-2
1.2 Open Research Issues C-2
1.3 Application Challenge C-3
1.4 Deliverables C-3
1.5 Related Work C-3
2 WebVM – Web Virtual Machine C-4
2.1 Computational Web Concepts C-4
2.2 HTTP+CGI – Minimal Model C-5
2.3 Management Modules C-5
2.4 High Performance WebVM C-6
2.5 Portable Module API C-7
2.6 Scripted Extensions C-7
3 WebFlow for Visual and Adaptive Programming C-7
3.1 Visual Authoring for the Web C-7
3.2 Navigation among Multiple Perspectives on HPC Data C-8
3.3 Extended Visual Programming Primitives for HPC C-8
4 Testbed Application – Adaptive Mesh Refinement (AMR) C-8
4.1 Applications of the Adaptive WebFlow C-8
4.2 Adaptive Mesh Refinement C-9
4.3 Workplan I: Identification and Extraction of AMR Modules and Dataflow C-9
4.4 Workplan II: New Modules for Visualization and I/O C-10
4.5 Workplan III: the AMR Dataflow Driver C-11
4.6 Summary: Implementing a WebFlow AMR Testbed C-12
5 Statement of Work C-12
5.1 Systems/Language Infrastructure C-12
5.2 Application Development C-13
5.3 Testbed Integration C-13

1 Introduction, Motivation and Overview

1.1 WebVM and WebFlow

We propose two cooperating software layers, WebVM and WebFlow, that will use familiar, broad-based Web technology to support high-level design and management of distributed High Performance Computing applications. The Web offers what HPCC has so far lacked: universal availability of standard client and server software, protocols and familiar visual user interfaces.

Adapting Web tools, protocols, and interfaces does not equate to chasing a computational fad. We believe that the explosive turnover in Web languages and protocols has begun to cool; HTML, Java, and VRML have now emerged as a relatively stable basis for long-term planning. Indeed, we argue that some basic lessons learned from the Web must soon be incorporated into HPC software, if we are to build, maintain, and use complex HPC systems effectively to serve national and defense needs.

Furthermore, our use of the term "Web" does not imply that systems which exploit Web technologies should be built atop the internet. In fact, use of Web technology for particular application domains and particular network systems has achieved great prominence recently. HPCC research "Intranets" consist of high-bandwidth, low-latency links engineered to maximize reliability and throughput. Our primary challenge is to retarget the Web-computing model (multi-server text, video, and executable software content) to meet the performance and reliability requirements of HPCC hardware and applications. Note that currently Java has low performance but we are not proposing here to use Java as a computational language but only for coordination and user interfaces where its capabilities are excellent and the performance issue is not critical.

We have already prototyped other aspects of the integration of HPCC with the Web. These prototypes include: the use of relational and object Web-linked databases to store Java and VRML to display information produced by large scale applications, a prototype of a PVM-linked collection of Web servers to provide a Web interface to HPF,(to get interface to HPF and the Pablo performance monitoring system), Java collaboratory technology, and a prototype of a Java visual editor for a distributed computing environment. In fact, our Web based computational model recently attracted the interest of the Binary Black Hole Grand Challenge community, weather prediction modeling systems [CAPS], and oil reservoir applications [Wheeler96] to implement robust AMR based PDE solvers.

For the Web paradigm to be made useful to HPC practitioners, we need to go beyond these initial tools to address some significant open research issues. These are not issues which are likely to be taken up by commercial software vendors in the foreseeable future since too many significant research issues must be solved before industry will invest in high-performance versions of Web software – an unorthodox approach to a relatively small market.

1.2 Open Research Issues

These challenging research issues include: (1) Integrated HPCC solutions that augment number crunching with collaboration, wide-area information systems, and distributed software development. (2) Scalable network wide virtual machine model. (3) Scalable I/O for heterogeneous distributed systems. (4) Interoperability of HPC components and collaborative coordination components. (5) Construction of a virtual, coherent presentation of performance visualization and real-time monitoring from monitoring tool outputs at the component level. (6) PSE little languages: very high level or domain specific languages for problem solving environments. (7) Object-based message passing versus MIME-based, and the need to develop HPC data object descriptor standards for CORBA/IDL. (8) A uniform, object-oriented HPC application development environment supporting portability and interoperability.

We propose the design and implementation of WebVM and WebFlow, bringing broad-based Web technology to HPCC contexts. WebVM defines a mesh of interacting Web servers, cooperating to match published hardware and software resources with browser clients in search of cycles and expertise. WebFlow adds a dataflow programming paradigm atop WebVM, complete with tools for visual authoring of computational dataflow graphs and integration of software modules coded in traditional HPC languages. Many of these concepts are inherited from previous dataflow systems such as AVS, Khoros, HenCE or CODE, and extended by Web-style publication, navigation and information processing capabilities. To demonstrate the unique benefits of our approach, we will test the WebVM/WebFlow

programming environment with a challenging application testbed: a PDE problem solving environment using Adaptive Mesh Refinement.

1.3 Application Challenge

AMR will demonstrate the challenges and merits of using the adaptive dataflow paradigm (WebFlow) to create a dynamic, self-adjusting network of workstations. Fortunately, distributed resource management is a primary exploratory focus at each of the proposer institutions: real-time Web infrastructure at Boston University, the MetaWeb study of cluster management systems at NPAC, and the WebMan project for network resource identification and administration at Cooperating Systems.

Modularity of the AMR algorithm promises an efficient implementation on an adaptive WebVM. Both external factors (dynamic system load balance) and internal factors (aggressive remapping of resolution at the algorithm level) combine in AMR. The same operations (unigrid hyperbolic and/or multigrid elliptic PDE solvers) are performed on all grids in the AMR hierarchy. These operations are local to a grid, and they are computationally expensive enough to amortize the overhead cost of intergrid communications (initialization of data on fine grids and restrictions). We will show that an adaptive WebVM platform for AMR computations can in fact be more responsive to requests for large and dynamically varying memory segments than an MPP with fixed memory per node.

1.4 Deliverables

WebFlow implementation will proceed in three stages. Stage 1 (months 1–7) combines conventional HTTP+CGI WebVM with a navigational front-end, hosted on a cluster of scientific workstations with a scalable high-speed interconnect. During stage 2 (months 8–12) we will introduce a visual program authoring system, based on Java applets and extended with HPC-specific visual program annotations, which will be coupled to the Stage 1 WebVM backend. Finally, Stage 3 boosts the performance of the WebVM layer by upgrading to Java-based, performance-optimized compute-servers (months 13–16) and advanced interactive visualization based on VRML 2.0, Java, and a DSI–like simulation framework (months 17–20).

The project will deliver an operational Web based AMR toolkit for PDE's and a more general purpose WebVM and WebFlow programming model for HPCC applications, ready to address other Grand Challenge and MetaComputing applications. Public release of software components takes place in each of the three stages, in months 7, 12, and 24, to give us time to integrate feedback from HPC programmers and endusers. Although a full problem solving environment is outside the scope of this proposal, our systems will probe many of the critical research issues which afflict HPCC applications development. WebFlow will integrate computing, parallel I/O, information processing, databases, visualization and scientific collaboration to answer those challenges.

To summarize, WebFlow will address issues (1), (3), (4), and (5) while WebVM will address issues (2). WebFlow will be an existence proof of a PSE little language, exposing all aspects of programming environment, tools, compiler, interpreter, and runtime systems issues. The WebFlow experience surely will illuminate approaches to the construction of full-blown problem solving environments, in whatever domain-specific languages (issue (6)). In addition, WebVM will address issue (7) by way of serving as a testbed for experimenting with much needed HPC object/data standards to address, for instance, interoperability of HPF and HPC++ components. Finally, WebVM/WebFlow is the first step towards addressing issue (8) by being an embryonic logically homogeneous, object-oriented, multi-threaded, interpretive environment that supports true portability and interoperability.

1.5 Related Work

Our unique choice of Web technologies to address HPCC software engineering and performance issues can be best understood by comparison with alternative approaches for widely distributed computing, starting with the Legion system [Grim95] from the University of Virginia. Legion-like systems have one obvious benefit: they simplify the task of low-level application construction by providing a simple, uniform API to all distributed components.

If Legion relies on drafting an army of resources, relying on universal conformity to enforce cooperation, then WebVM can be thought of as the "guerilla warfare" approach to high performance computing. The WebVM approach assumes that performance and administrative asymmetries will inevitably be present in any nontrivial Intranet. Instead of hiding them, WebVM exposes these asymmetries to the control of the low-level application programmer,

relying on designers to appropriately rewrap, Web-publish, and hyperlink the available software components on each machine. We believe that the trade-offs in Legion and WebVM/Flow are distinct and complementary so that it is important to investigate both

For the WebVM approach to succeed, we will need to provide cluster management support to assure fault tolerance, robustness, distributed control and the adaptive load balancing in such a diverse collection of hosts. We are evaluating several promising cadidates such as the Condor system [Wisc96] from Univ. of Wisconsin, algorithms for distributed contol of ATM networks [Tuner95] and other cluster computing frameworks collected in our recent review [Baker95]. Another important issue associated with the distributed cluster management to be addressed is distributed system security; specifically, the ability of intruders to detect and exploit design or implementation imperfections of the management layer. Here we plan again to benefit from the existing research such as pursed in the COAST project [Spaff95] at the Purdue University.

Our approach to network computing, exposing Web-based software publication and coordination, is synergistic with some new HPCC Internet initiatives such as Caltech Infopsheres [Chari96] (one of the communication models for our Java co-server layer — see Section 2.4) and complementary to some other ongoing projects in the HPCC community such as Netsolve [DoCa96] that focuses on base high performance remote computational services, or I-WAY/Globus [FoKe96a,b,c] that focuses on raw ATM performance. We believe that the optimal computing environment for the Internet will emerge by combining approaches driven by performance and usability. In fact, we are already collaborating with Argonne National Laboratory on combining their broadband video technologies with our Web based electronic presentation technologies in the science collaboratory project that will bring us closer to the I-WAY community and the associated software technologies.

On top of this raw multiserver WebVM layer, we then add WebFlow, our higher-level programming environment for Web-based dataflow. The objective is to allow complex low-level applications be coded once atop the raw WebVM, by the appropriate domain experts, and reused many times by application endusers through the Web's familiar point-and-click interface. In other words, WebFlow gives WebVM endusers the benefits of both worlds: a simple user interface and the performance benefits of a thin, flexible software layer per host. We believe the light-weight, low overhead WebVM, coupled with an application such as WebFlow, provides a reasonable ease-of-use/performance tradeoff for HPC applications. In the WebFlow area, we will again benefit from the previous accomplishments of the HPCC community, represented by visual authoring systems such as HeNCE [Dong94] or CODE [Browne92]. In fact, the recent version of CODE already addresses the adaptive computational trees; we will reuse this expertise in the our AMR visual programming frontend, discussed in Chapter 4.

2 WebVM – Web Virtual Machine

2.1 Computational Web Concepts

The Common Gateway Interface (CGI) protocol currently allows Web clients to access executable content (software) published by Web servers. Typical applications are dynamic document generation, processing of form input, and accessing database servers. CGI processes are also used as arbitrarily complex filters for data fetched from the client side; popular examples include AutoCAD DXF to VRML file convertors, remote image editing and improvement services, and remote VRML authoring kits. WebFlow takes this progression to the next logical step, by using the dataflow model to chain together collections of these network services.

We first demonstrated the feasibility of distributed computation atop Web standards with a simple application - parallel search [Veli95] - built atop a mesh of WebTools [Furm94] servers. A more advanced WebWork application in the area of large–integer factorization within the RSA factoring challenge was presented at Supercomputing 95 and awarded within the 3rd Annual High Performance Computing Challenge Awards contest. [Cowie95].

Meanwhile, AVS users have created a community-owned library of visualization modules available from the International AVS center, making it simple to develop AVS applications. We showed [CFLH96] that distributed dataflow implementation of a collection of these modules allows the amortization of the costs of intermodule communication: a rudimentary form of coarse-grain parallelism.

To take WebVM/WebFlow beyond the simple multimodule aggregation provided by systems such as AVS, however, we must address some significant open research issues. Simple standards for module connectivity are

already commercially available; the key for HPC is to address the hard problems of HPC software: support for adaptivity and dynamic load balance, provisions for fault tolerance, and interfaces to standard HPC programming languages, tools, and low-latency local communications.

The following sections describe the sequence of fundamental research advances that will be required to reach these goals. Section 2.2 describes "minimal WebVM" — one software module (the "Flow Executive") that implements the simplest dataflow protocol atop the familiar HTTP and CGI protocols. This stage of development is analogous to infrastructure for raw message passing in a non-dataflow distributed application.

Section 2.3 then plugs in additional software modules: the Link Manager for updating the dataflow connectivity database, the Configuration Manager to publish an interface to that database, and a Resource Manager to support checkpointing, fault identification, rollback, and static and dynamic load balancing. This completes the description of the basic WebFlow software architecture.

Section 2.4 describes some additional engineering techniques for performance augmentation, based on CGI servers that exploit lightweight multiprocessing and improved cache management. These improvements will widen the 'flexibility gap' between our lightweight WebVM approach and the heavy virtual-OS approach of Legion-like systems. Section 2.5 details our plan for deriving a portable API for software modules based on a "little language" approach, and section 2.6 shows how this software engineering technique ultimately avoids the "binding explosion" common to thick-layer multilanguage approaches to software module integration.

2.2 HTTP+CGI – Minimal Model

The simplest prototypes of WebVM, which we have already constructed, consist of collections of Web servers, one per node of a workstation cluster. Each server publishes two sets of CGI scripts ('modules'): one set of WebVM system services and another set of user-provided application modules. Each module is written in a traditional HPC language (e.g., Fortran 90 or HPF) and may use PVM or MPI internally for high-performance intra-module communication. Upon activation as a CGI script, each system or application module receives a set of input MIME objects, performs an appropriate local computation, and returns a set of output MIME objects. Both input and output objects are written to disk and are URL-addressable; i.e. they reside as MIME documents in a server document tree.

One special WebVM system module, the Flow Executive, carries out server-to-server object transfer via HTTP. Fig. 1 illustrates the steps involved in transferring the output object $O_1{}^{(1)}$ created by module M_1 from the server S_1 document tree to the server S_2 document tree, where it is stored as the input object $I_1{}^{(2)}$, ready to be used by the module M_2. The transfer starts after the module M_1 has finished its computation and writes $O_1{}^{(1)}$ to the local disk. After completing the write action, M_1 opens up a connection to S_2 and sends a POST message to the Flow Executive F_2, passing the current URL of stored $O_1{}^{(1)}$. F_2 sends the GET message to S_1, retrieves $O_1{}^{(1)}$, and stores it as $I_1{}^{(2)}$. Next, F_2 starts module M_2 in the background and exits. At this point, M_1 can also exit and all connections between servers S_1 and S_2 are closed. Module M_2 reads its input files $I_1{}^{(2)}$, $I_2{}^{(2)}$, performs local computation, writes its output files $O_1(2)$, $O_2{}^{(2)}$, and notifies the Flow Executive processes in the destination servers.

Just as familiar Web servers have no problem serving multiple documents to multiple browsing clients simultaneously, our WebVM servers can participate in multiple dataflow graphs (called "compute-webs"). Because ongoing multiserver dataflow adds some state to previously stateless Web servers, temporary configuration files in each server track persistent connectivity information for each compute-web.

Dataflow is initiated when a browser accesses the "front door" module of a compute-web, (e.g., M_1 in Fig.1), as a CGI script. Other modules (M-processes) are activated whenever the state of any of their input objects changes, and F-processes are activated as object transfer requests are issued by M-processes. A prototype of this minimal model for server-to-server communication has been built at NPAC [Haupt96].

2.3 Management Modules

To construct a truly usable distributed computing environment for High Performance Computing, we propose an additional set of WebVM per-node system services alongside the basic Flow Executive. New modules can be wired into and removed from a live compute-web by consulting the local Link Manager service, which updates the compute-web configuration files.

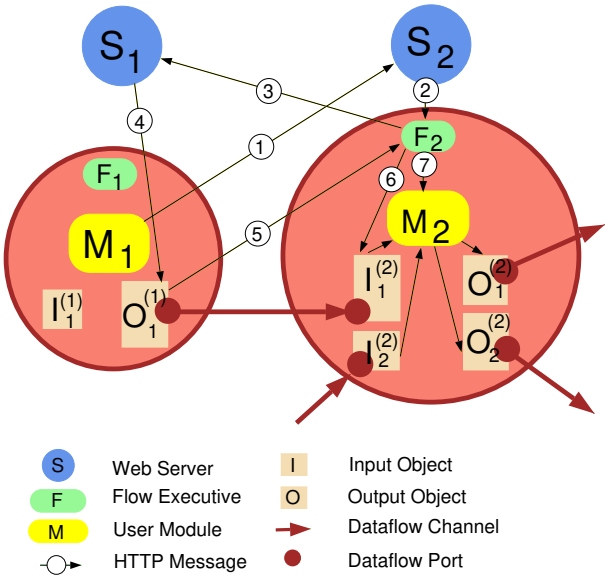

Fig. 1: Minimal model for HTTP+CGI based Server-to-Server Communication discussed in Section 2.2

The Configuration Manager assures atomicity and concurrency control in the multi-user interactive environment by implementing appropriate locking and access protocols to the compute-web configuration files. Where preliminary prototypes use a lighter weight DBMS or even raw text files to store this configuration information, complex WebVM systems will use high-performance relational databases [oracle92].

Finally, a Resource Manager module will provide fault tolerance, static resource allocation, and dynamic load balance via module migration. Replication and caching of the document tree that stores intermediate dataflow results will allow the Resource Manager to support checkpointing, fault identification, and computational rollback for very large, lengthy distributed computations.

For this, the final and most complex WebVM system service, we combine particular strengths of each proposer institution. Ongoing work at Boston University [RWC96] focuses on support for real-time Web transactions, advanced document caching strategies, and Web-based resource management systems. NPAC is currently exploring several existing Cluster Management Systems (CMS) [Baker95] and selecting the most adequate solutions for adoption into the WebVM. Cooperating Systems has an ongoing project, WebMan [Webman96], to produce Web-based tools for network resource identification and management. WebVM's modular software architecture will absorb the benefits of each of these three efforts as they come online.

2.4 High Performance WebVM

As previously noted, preliminary WebVM prototypes have been built atop stateless Web servers, augmented with minimal state in the form of MIME files cached in the local document tree. This has the downside of requiring new socket connections for each dataflow channel, and a new process for each dataflow channel manager, resulting in high latency. New HPCC-friendly Web contributions from the HPCC community (Netsolve, Globus/I-WAY, Infospheres) open up new and higher performance implementation avenues for the WebVM architecture.

One higher performance implementation of the dataflow of Fig.1 could require auxiliary servers per node, cooperating with Web servers to improve caching strategies. Another improvement would utilize object-oriented, multithreaded CGI servers, replacing MIME files with live object instances and full CGI processes with spun threads. The current rush to produce a standard Java-CORBA binding will promote this trend toward lightweight collections of threaded, interoperable collections of distributed Web objects. Each of these solutions would narrow the communications performance gap between the outer, coarse-grain coordination layer between modules (Web-based dataflow) and the inner, fine-grain traditional message passing exploited within each module.

Our proposed WebVM servers would start with a minimal Java server per node, which would grow dynamically by downloading additional system and application classes. Performance-critical WebVM services would will be systematically migrated from our initial HTTP+CGI servers to the higher performance distributed Java layer.

2.5 Portable Module API

Because Web technologies change rapidly, our proposed WebVM software architecture will be maximally platform and vendor independent. Each W3C compliant Web server in the world should be able to participate in high performance computations, simply by installing the appropriate set of WebVM CGI scripts.

The Web Virtual Machine concept standardizes a module-level programmer interface while hiding module-internal details of the WebVM implementation. WebVM users develop application-specific modules using appropriate HPC language(s) and protocols (HPF, MPI), and leave coarse-grained dataflow to WebVM system services. The API for module interaction is profoundly simple, consisting of just one type (named dataflow ports, the heavy dot in Fig.1) and two function calls: inp_handle = port_read(port_name) and port_write(port_name,out_handle). For example, module M1 would execute port_read twice to receive handles for input objects $I_1^{(2)}$, $I_2^{(2)}$, perform the computation, then execute port_write twice to pass handles for the output objects $O_1^{(2)}$, $O_2^{(2)}$ to the system layer, and exit. This minimal API is standard across all WebVM implementations.

2.6 Scripted Extensions

Conventional dataflow is most useful for topologically static distributed applications. However, several relevant scientific computing and information processing problems require dynamically varying topologies. For example AMR algorithms for time—dependent PDEs dynamically generate new transient tasks for refined grids, and a WebAMR programmer would like to distribute such processes efficiently over the WebVM nodes.

One possible solution involves extending the module API by domain specific calls such as grid_refine etc. However, API extension would mix domain specific and generic WebVM components and result in a complexity explosion, since each domain-specific call needs to be associated with multi-language binders for all languages supported by WebVM. We chose instead a more elegant and problem-independent approach based on scripted extensions of the base WebVM model.

In terms of the minimal WebVM model and its management modules, we introduce another system process called Script Executive, and one more routine in the module API – script_execute(language, string) — which passes scripts in the application-specific little language to the Script Executive for immediate execution/interpretation. Script Executive will offer a default set of commands such as creating, destroying or modifying the compute-web topologies, as well as mechanisms for installing new application specific commands by users or their domain system administrators.

Script Executive builds its functionality on top of the base WebVM layer, including server-to-server message passing, and hence it can also exploit the distributed interpretation capabilities. For example, a user's command to create a new node in the current compute-web can be implemented with built-in dynamic load balancing: Script Executive can issue a status request to Configuration Managers, select the optimal node based on this information, and finally send the module placement request to the Link Manager in this node. An alternative and often more efficient strategy for load balancing could use periodically refreshed information maintained by local Resource Managers but the scripting channel will offer more direct insight into the real-time operation of WebVM and a platform for refining the existent and prototyping new commands in the WebVM instruction set. System and user strategies can also be more easily integrated in the scripted mode. For example, applications may provide a special handler module to refine or even override the default node placement process and pass the URL of such a handler to the Script Executive as part of the command script.

3 WebFlow for Visual and Adaptive Programming

3.1 Visual Authoring for the Web

Several high level programming paradigms can be implemented on top of the basic software architecture discussed in Chapter 2. WebVM and WebFlow both focus on component technologies; most of the hard unsolved

research issues in distributed computing involve the component integration aspect. The implementation of individual HPC modules within WebFlow takes advantage of existing languages and tools: HPF for computation, MPI for local communication, AVS for visualization. Visual programming is therefore a particularly attractive candidate for the final link: an interactive environment for coarse grain coordination of distributed HPC modules.

Previous experience with AVS and Khoros suggests that a visual Web programming paradigm would help tame the complexity of the multi-language Web. Today, a Web developer must be a proficient HTML, Perl, Java, JavaScript, LiveWire and VRML programmer. WebFlow's visual programming tools will hide this complexity behind intuitive visual icons and click-and-drag design metaphors.

3.2 Navigation among Multiple Perspectives on HPC Data

Advanced computing in science and engineering usually refers to some complex space, with the individual modules operating in physical or functional subspaces. This important feature of advanced computing has not been satisfactorily exploited by existing HPC software tools.

By contrast, navigability is the cornerstone of the Web. WebFlow tools will allow the enduser to navigate among multiple perspectives on HPC data by using a single dataflow specification to link different software modules. Instantiated with one set of AVS modules, the WebFlow application's dataflow pattern guides advanced visualization. Another set of modules might highlight resource utilization and load imbalance. Still another would flag software faults and detect deadlock conditions, and so forth. Users will follow standard Web hyperlinks to step from one perspective to the next, and from one region of the complex HPC application space to another.

3.3 Extended Visual Programming Primitives for HPC

The first commercial tools in the area of visual Web programming are already available (e.g. JFactory for Java software engineering). NPAC has already prototyped a visual editor based on Java applets [Lasz96]. Other tools for visual programming will follow from JavaSoft and other commercial interests.

Again, however, these first easy steps will not meet the needs of HPC application designers without some additional hard work. WebFlow will therefore go beyond the obvious (and traditional) visual programming primitives: visual metaphors for editing, hierarchical composition, abstraction, instantiation, and so forth.

We propose a set of visual programming annotations for high performance codes, designed to support the special mapping needs of adaptive HPC applications onto dynamic meshes of resources. For example, WebFlow users may click to annotate large data structures with "split points" during the visualization process. Human visual intuition will help locate and designate internal surfaces which minimize communication between application components separated by high-latency WebVM links. Other tasks which can benefit from visual programming annotations include interactive bandwidth allocation, control over storage replication and I/O striping, and incremental remapping of the dataflow graph onto computational resources.

Adaptive Mesh Refinement is an ideal test application. The components utilized in AMR can be treated as separate modules which can be combined in a data flow fashion into one large module - a single node in a distributed WebVM machine. In addition to built-in low level interactive coordination and scheduling functionality, a complete WebFlow AMR system will incorporate entire multigrid solvers, I/O and visualization modules.

4 Testbed Application – Adaptive Mesh Refinement (AMR)

4.1 Applications of the Adaptive WebFlow

Adaptive Mesh Refinement for PDE solvers is an HPC application area of immediate practical importance to efforts ranging from multiscale algorithms from fundamental physics (Binary Black Holes[BBHGC]) to practical engineering (multiresolution CFD simulations as pioneered by NPSS and NASA Lewis[NASA]). In this section we describe the application of WebVM/WebFlow to this family of critical interdisciplinary HPC applications. The ability to migrate AMR software, now the exclusive domain of expensive MPPs, to a dynamically configured, distributed memory environment with a point-and-click approach to AMR program composition would be a major achievement.

In particular we will focus on the use of AMR for hyperbolic and elliptic partial differential equations. The WebFlow AMR Testbed will support custom transformation of an existing stable unigrid solver to a stable multigrid solver. WebFlow is an ideal platform for this challenge, able to satisfy AMR's memory- and CPU-intensive distributed processing requirements while offering the enduser a familiar GUI interface to dataflow computation over libraries of reusable AMR modules.

WebFlow AMR construction will start from a basis of existing PDE solvers developed by ongoing Grand Challenge research and other projects. We will modularize those systems (extracting functional components such as interpolators and Poisson solvers), formalize the patterns of dataflow between them in AMR applications, and expose the coarse-grain concurrency latent in such systems. Through the familiar Web browser interface, using a visual editor we provide as a Java applet, WebFlow users will point-and-click to recombine existing AMR-based PDE solver modules to implement a convergent and stable multigrid code.

There are three primary components of the WebFlow AMR workplan: (1) the modularization of existing AMR PDE solvers into a collection of reusable modules, (2) the design of a new family of visualization and I/O modules to support multilevel evolution and 3-D visualization packages such as VRML 2.0, and (3) the all-important AMR dataflow driver that maintains a dynamical adaptive grid hierarchy and distributes workload across a WebVM.

In the following sections, we first give a brief description of AMR techniques, omitting for brevity the full description of the algorithms used [Berger84]. We then explore each of the three WebFlow AMR workplan components in more detail. Finally, we summarize by describing our implementation plans for a WebFlow AMR Testbed.

4.2 Adaptive Mesh Refinement

To address the problem of insufficient resources to solve systems of hyperbolic PDEs with required precision, Berger and Oliger introduced a new technique of integration, commonly referred to as Adaptive Mesh Refinement (AMR)[Berger84]. This technique allows one to compute the solution with a variable resolution, adaptively concentrating computational effort on selected computational subdomains, corresponding to a variable scale of the underlying physical processes. This is achieved by tracking regions in the domain that require additional resolution and dynamically overlaying finer grids over these regions. AMR techniques start with a base coarse grid with minimum acceptable resolution that covers the entire computational domain. As the solution progresses, regions with unacceptably large truncation errors are tagged and finer grids are overlayed on the tagged regions of the coarse grid. Refinement proceeds recursively so that regions on the finer grid requiring more resolution are similarly tagged and even finer grids are overlayed on these regions. The resulting grid structure is a dynamic adaptive grid hierarchy.

The same base unigrid PDE solver – a grid update operator – is applied to each grid on all levels of refinement. The Berger-Oliger algorithm defines the sequence of grid integration (see Fig. 2) and requirements for data exchange between grids. Conversion from a unigrid mode to AMR thus does not involve any modifications of the original unigrid PDE solver. Instead, it becomes a module in adaptive dataflow computations, with data dependencies defined by the Berger-Oliger algorithm.

Our expertise in AMR comes mainly from the research done in Numerical Relativity. We started working from a one dimensional sequential implementation to investigate critical phenomena in black hole formation [Chop94]. This experience became a foundation for a general purpose 3-dimensional, data parallel implementations in HPF[Hk96] and in C++/MPI (DAGH) [Par95]. Our experience from MLAT is based on work from Klasky[Kla94].

4.3 Workplan I: Identification and Extraction of AMR Modules and Dataflow

The Berger-Oliger algorithm can be divided into several relatively separate components – modules. These components fall into three categories: operators that act on a single grid, operators that act between two grids, and operators that modify the grid hierarchy. This modularity makes Berger-Oliger an excellent target for dataflow implementation.

The first module to be built is the update operator that moves the solution on the grid from time t to $t + \Delta t$, where the size of Δt is proportional to the spatial resolution on given refinement level in order to satisfy the CFL condition. The update of the boundary conditions, if these change in time, can be included here, or they can be

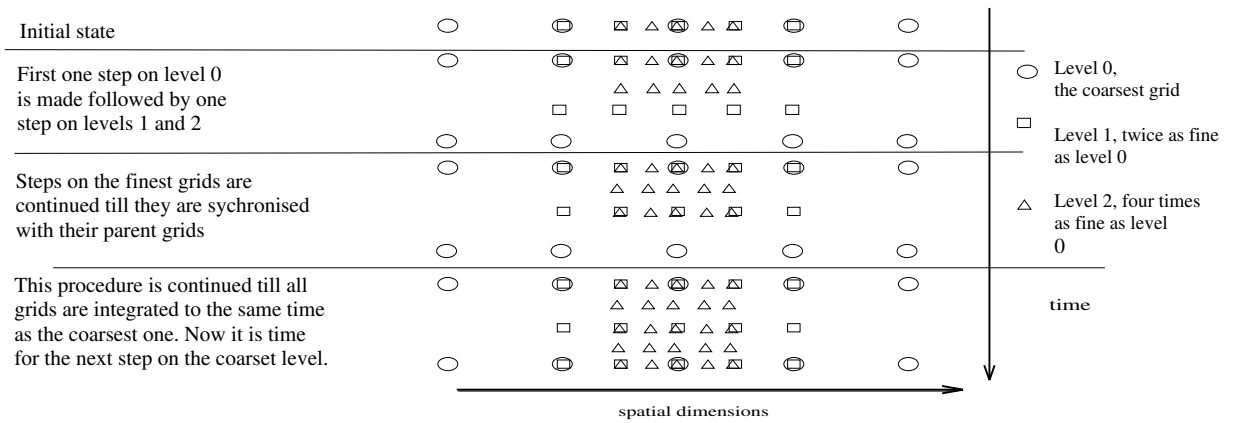

Fig. 2: The sequence of grid integrations according to the Berger-Oliger algorithm. Every p steps regridding take place (not shown).

implemented as a separate module. The next module is the truncation error estimator, which is usually based on the Richardson extrapolation [Rich10]. The output of this module is then processed by a clustering routine which transforms a irregularly distributed set of flagged points into Cartesian grids [Berger91, Chriso]. At this moment the actual regridding takes place, namely the old grid hierarchy is replaced by a new one, in other words, the data flow scheme is updated. Once the new structure is created, new grids are initialized. This is done by spatial and temporal interpolations [Kla95] that can be encapsulated as stand alone modules. Finally, there is a need for a module that performs flux corrections [Berger94]. This insures conservation at grid interfaces by modifying the coarse grid solution for coarse cells that are adjacent to a grid. In addition, support for I/O and visualization must be incorporated into this scheme; section 4.4 describes some new modules we will design for this purpose.

The dataflow paradigm also simplifies development of even more complex PDE systems: systems of coupled hyperbolic and elliptic PDEs. Elliptic solvers, to speedup convergence also employ a multilevel integration scheme. Contrary to AMR, the multigrid methods [Brandt] uses coarsening rather than refinement of grids. There are two basic ways to incorporate multi-grid into our AMR scheme.

The first method assumes that on every time step when we must solve an elliptic problem then we multi-grid off of each grid in the distributed adaptive grid hierarchy separately. This then involves adding data structures to handle the coarse grids that are used for the coarse grid corrections in the multi-grid algorithm. Each fine grid problem is viewed as solving a Dirichlet boundary problem, where boundaries remain unchanged in the elliptic problem. This approach invites a hierarchical dataflow model: each AMR module exhibits a fine structure with an internal dataflow which can be visually programmed. The other method involves solving the elliptic system over the entire hierarchy. This is a Multi Level Adaptive Technique (MLAT). In this approach, additional grids might have to be created in order to accelerate the convergence of the MLAT solver. A more detailed discussion of MLAT can be found in [Kla94]. MLAT will be used as our testbed for static AMR implementations. The MLAT technique is more challenging to implement in a dataflow paradigm and certainly requires some research in order to define optimal granularity of modules and their interactions with the flow manager.

At the end of this workplan phase, we will have defined appropriate MIME data types (e.g., GRID, including a grid id, bounding box, spatial and temporal resolution, and actual functions defined on the grid). We will also have identified and broken out an appropriate family of modules implementing AMR operators in a mix of traditional HPC languages: HPF and C++/MPI, along with the static and adaptive dataflow patterns (exchanging data of type GRID) suggested by Berger-Oliger. Conformance to traditional HPC language standards is critical, to promote migration of codes between WebVM and MPP environments such as DAGH.

4.4 Workplan II: New Modules for Visualization and I/O

Designing in adequate support for distribute high-performance I/O and visualization are critical to the acceptance of WebFlow methods for AMR. Because evolution data must be written every several time steps, resulting in

terabytes of output, we will construct distributed I/O modules that interact with high-performance remote database servers from Illustra/Oracle [citeoracle] to satisfy the I/O requirements of the distributed collection of AMR solver modules. Storing the data from various runs is another task which will require extensive relational or object DBMS support.

The combination of human interface issues with distributed processing challenges has made visualization of a multilevel evolution an active area of research. Our pre-WebFlow research implementations wrote a new HDF (binary compatible from machine to machine) file for each grid used in the calculation. We have also worked on the developement of a client server data visualization package for one dimensional AMR along with M. Choptuik[Chpser]. To construct an adaptive distributed visualization scheme, however, we will need to build some additional WebFlow modules for AMR visualization, which can be instantiated on-the-fly in the process of grid creation.

Unlike AVS, which requires that the user define a static module map, WebFlow AMR will support incremental remapping of the computation and the visualization modules aligned to it. MIME-format AMR dataflow temporaries stored on local server disk will provide the raw data for visualization; the GRID document types we define will include enough information to identify spatial and temporal position of the grid.

By designing new visualization modules that speak VRML 2.0, we can support 3D visualization of data from each grid, linked by standard Web hyperlinks for navigation. Other modules (written in Java) will allow the user to click on the 3D image to obtain a series of two dimensional slices, or one dimensional line plots [VH,NCSA]. WebFlow enables Web-publication and long-distance collaboration by making these images available through the standard Web browser interface. Our VRML modules will allow multiple researchers to interact with the AMR visualization, viewing and manipulating a common scene in virtual reality to help the scientific process of extracting meaning from the data.

4.5 Workplan III: the AMR Dataflow Driver

AMR techniques allow for a significant reduction of the required resources by performing computations with variable resolution, to obtain accurate solutions in critical regions of the computational domain. The demands for processing power, even with AMR, far exceed the current high-end workstations when performing multidimensional calculations. Therefore the current implementations of the AMR are targeted mainly to massively parallel computers with hundreds on computational nodes, or vector supercomputers.

Existing MPP implementations typically implement unigrid solvers in data parallel, either using high performance message passing libraries such as MPI, or data parallel languages such as HPF. Advances in compiler technology, in particular C++ and Fortran 90/HPF made the problem of dynamical memory management easier to code efficiently, and in fact port it to the parallel distributed memory systems. However, distributed implementations of AMR add an additional level of complication to the already complex AMR algorithm. Challenges include load balancing, minimizing communication costs, handling irregular boundary conditions and overall parallel efficiency.

In order to use AMR solvers on adaptive WebVM we need an approach that will eliminate critical dependency on the latency of the system. A natural solution is to expose task parallelism rather than data parallelism, and rely on dataflow techniques to synchronize between nodes in the regridding and restriction phases.

The modular structure of the Berger-Oliger algorithm makes it possible to implement AMR in a classical data flow paradigm on WebVM. At any time instance we have a set of modules, most of which are identical unigrid update operators. The data dependencies between the modules are uniquely defined by the Berger-Oliger algorithm. These intermodule dependencies capture all the coarse-grained data dependence in AMR, since each node is assigned a whole grid, and all computations are performed locally, independent of computations executed on other processors except at the designated intermodule synchronization points.

All modules involved in the regridding have critical importance for both efficiency and stability of the PDE integration. The optimal choice of parameters such as the threshold value of the truncation error or clustering efficiency as well as particular algorithms for clustering or interpolation is application dependent. Moreover, initialization of the fine grid includes imposing boundary conditions on the newly created grids. This is obviously correlated with the integration method adopted in the update module. The WebFlow visual editor allows users to choose the modules which best fit their application and sometimes replace modules by their own version.

We will construct an extended version of the WebVM Flow Executive, the AMR Flow Executive, to add AMR-specific adaptivity to the data flow. This extended WebVM service will allow the computational modules at the regridding phase to interact directly with the extended Flow Executive in order to dynamically add and remove modules.

The structure of data dependencies (order of grid updating) suggests several strategies for optimizing the implementation of the AMR Flow Executive. For example, a grid on a coarser level cannot be updated before restrictions from finer levels are completed. The fine levels, even though they span smaller physical domains, may be comparable in size to the parent grid, because of their greater density of mesh points. Moreover, since the refinement is made in both spatial and temporal dimensions, the actual computational task on a fine grid may be larger than that on the coarse grid. All this indicates that a naive distribution of one grid per node may be in practice very inefficient. Some load balancing will be certainly necessary to increase parallel efficiency of the system and increase throughput. Therefore, it may be necessary to assign processing of more than one grid to a single processor. It can then be possible to reuse ideas implemented in DAGH for load balancing imposing the constraint that no grid can be split among processors [Par95].

4.6 Summary: Implementing a WebFlow AMR Testbed

Our workplan separates the design of AMR as a modular dataflow system from the specific binding of that dataflow system to our WebVM/WebFlow implementation. Mating one to the other is the subject of our final goal: constructing a WebFlow AMR Testbed based first on a workstation LAN, and then on a network of shared memory SMPs interconnected by high performance, low latency switches. In the next section, we describe the sequence of milestones and software releases required to achieve this goal.

5 Statement of Work

This proposal concentrates on designing and prototyping new technologies to support scientific and engineering computations using composition and coordination technologies borrowed from the World-Wide Web. We will demonstrate the feasibility of implementing a Web Virtual Machine to carry adaptive dataflow computations, and construct a visually-oriented dataflow programming environment, WebFlow, for high performance computation. The 24-month project divides the challenges of developing HPC WebVM/WebFlow into three fronts: Systems/Language, Applications, and Testbed.

5.1 Systems/Language Infrastructure

The first, Systems/Language, focuses on designing the program interfaces, scripting languages, and distributed dataflow infrastructure that will be made available to WebFlow applications designers. Within this subproject, three primary deliverables emerge, reusable by all future WebFlow HPC applications :

1. a complete WebVM/WebFlow system, with Link Manager, Configuration Manager, and Resource Manager, supporting navigation among multiple perspectives of the same distributed computation. A basic distributed dataflow infrastructure with only one service, the Flow Executive, will be provided before the project begins.
2. "little language" interpreter, in the form of a Script Executive Module. A minimal model for augmenting compute-web topology, based on suitable extensions of popular server-side interpreters such as Perl or Livewire, will be provided on day 1.
3. visual editor extensions for HPC WebFlow: A new set of visual annotations for human-in-the-loop control over dataflow graph partitioning, distributed resource mapping, heterogeneous I/O. A basic Java-based visual editor, implementing familiar graphical programming primitives for composition and control, will be provided on day 1.

This software will be released to the public in three stages to elicit feedback. The first ("Bronze") release will contain the first draft of the extended WebVM infrastructure: compute-webs, Link Manager, and Configuration manager services. The second ("Silver") release will add the Script Executive's support for scripting language extensions, and a Java-based visual editor with HPC annotations. The final ("Gold") release will include navigation among multiple perspectives, supporting collaborative interactive visualization in virtual reality (VRML 2.0).

5.2 Application Development

The second development front, Applications, demonstrates how to develop one practical HPC application (PDE/AMR) as a distributed collection of cooperating software modules. This exploration of the structure of a distributed AMR application will provide critical feedback for the adequacy of our proposed WebFlow API. However, AMR is at the same time representative of a much larger family of complex scientific applications which can take advantage of the WebFlow infrastructure.

5.3 Testbed Integration

Finally, the Testbed phase mates the AMR application to the system/language infrastructure, resulting in a complete prototype which can be emulated by future designers of HPC WebFlow applications. Performance benchmarks, demonstrations of ease-of-use and programmability, and promotion of the testbed to a developing WebFlow user community will all be of vital importance to the acceptance of Web technologies serving HPC interests.

Month	System/Language	Applications	Testbed
0	At project inception: availability of zeroth-order Web tool infrastructure already prototyped. Java-applet extensible visual editor with standard primitives built-in, WebVM prototype with basic dataflow support.		
1-2	Development of the simplest dataflow API, and bindings to C/C++, Fortran90, Perl, Java, and VRML. Preliminary design of a set of visual program annotations fro HPC. Release basic WebVM and dataflow API to Testbed team.	Analysis of the AMR and MLAT algorithms in order to define an API. Feedback to the systems group based on algorithms requirements.	Extend the current proof-of-the-concept WebVM layer operational at NPAC to support API design testing.
3-5	Extend infrastructure (configuration files, Link Manager, Configuration manager) to support multiple adaptive compute-webs per WebVM server. Extend visual editor to include support for final design of visual program annotations.	Select a representative suite of PDE solvers for the testbed. The suite must contain also simple codes, such as wave equations (1d and 3d) and Laplace's equation for early tests.	Develop more modules according to the API specifications to test early implementation of WebVM. Develop simple visualization modules based on VRML 1.0
6-7	Deliver extended system infrastructure to the Testbed team; develop and integrate Resource Manager module. First public release ("Bronze") of basic WebVM system software for comment.	Generate reference results which will be used for validation of the testbed. Collect elements of the MLAT/AMR support library (interpolators, clustering routines, etc)	Enhance visualization modules (in Java and VRML) in order to test static WebFlow.

8-10	Develop the underlying dataflow mechanism to implement the visual program annotations designed in months 1-2 and integrated into the editor in months 3-5. Integrate the visual editor with the WebVM infrastructure.	Select and test components of the MLAT support library that will be converted to the WebVM modules.	Start WebFlow implementation of MLAT applications, since this is a static implemention.
11-12	Design and implement Script Executive module for interpretation of application-specific "little languages"; integrate with WebVM infrastructure by migrating initial interpreter prototype to the evolving highier-performance Java Webserver layer. Second public release ("Silver") of WebVM system software.	Select and test components of the AMR support library that will be converted to the WebVM modules.	Test static DataFlow using MLAT. Test and improve visualization modules. Implement AMR modules.
13-16	Begin migration of WebVM modules to highier-performance Java-based servers based on lightweight multithreading and CORBA. Preliminary design infrastructure for interactive management of multiple visualization perspectives within a single compute web.	Work on convergence and stability of codes. Exchange experience with other groups.	Develop scripts to control dynamic DataFlow as required by AMR.
17-20	Implement WebVM modules for collaborative interactive visualization using Java and VRML 2.0.	Test and validate MLAT and AMR implementation.	Benchmark different variants of the WebVM. Implement new visualization features.
21-24	Third and final public release ("Gold") of WebVM system software.	Publish AMR results using visualization tools on the WWW	Package, document and disseminate the testbed software.

FACILITIES, EQUIPMENT AND OTHER RESOURCES

The Northeast Parallel Architectures Center (NPAC) is a research and development center focusing on high performance computing and communications. Established at Syracuse University in 1987, the center is directed by Geoffrey C. Fox, a pioneer in the development and application of parallel computers, who assumed NPAC leadership in 1990. NPAC has a strong emphasis on computational science applications where one seeks to obtain "real" solutions to "real problems." NPAC InfoMall technology transfer program puts high-performance computing and communications (HPCC) to work in industry. Other major projects include research and development in the ears of parallel languages and parallel compilers, including work on High Performance Fortran (HPF) - a standardized parallel version of Fortran' distributed computing; parallel database technology; integration of relational databases with the NII; parallel algorithms; distributed and cluster computing; and networked digital multimedia applications providing Information, Video, Imagery, and Simulation on demand.

Major Equipment

In this proposal we plan to use NPAC's high-end workstations to implement and design the WebVM. NPAC has over 100 workstations which we will use to develop this technology. NPAC's SP2 will also be used in this project. Here we will be able to compare our Web implementation of AMR with our previous MPI implementation of AMR. No other/new computer resources will be required for this project.

BIBLIOGRAPHY

[Baker95] M. Baker, "Cluster Computing Review", November 1995
http://www.npac.syr.edu/techreports/html/0700/abs-0748.html

[BBHGC96] Binary Black Hole Grand Challenge Alliance,
http://www.npac.syr.edu/projects/bbh

[Berger84] Berger, and Oliger, J. Comp. Phys, 53, pp 484 (1984).

[Berger91] Berger, and Rigoutsos, IEEE Trans. on systems, man, and cybernetics **21** 1278 (1991).

[Brandt] Brandt, Math. Comput. ***31***, 333 (1977).

[Browne92] J.C. Browne, "CODE – The Computationally Oriented Display Environment", University of Texas at Austin, 1992
http://www.cs.utexas.edu/users/code

[CAPS] Xue, Droegemeier, and Woodward, "Simulation of tornado vortices within a supercell storm using adaptive grid refinement technique", Preprints, 17th Conf. on Severe Local Storms, St. Louis, MO, Amer. Meteor. Soc., pp 362–365.

[ChaRi96] Mani Chandy, Adam Rifkin, "The Caltech Infospheres Project"
http://www.cs.caltech.edu/~adam/CALTECH/infospheres.html

[Chriso] A three dimensional implementation of a clustering routine
http://www.npac.syr.edu/projects/bbh/PAMRSL/clustering/cluster.html

[Chop94]An implementation of the Berger & Oliger mesh refinement algorithm for the wave equation in spherical symmetry.
http://godel.ph.utexas.edu:80/Ftp/ad/doc/ad.ps

[Chpser] A one dimensional client server visualization package for AMR applications
ftp://helmholtz.ph.utexas.edu/pub/sgi/mwc/bin/

[Cowie95] J. Cowie et al., "RSA130 Factoring by Web"
http://www.npac.syr.edu/factoring.html

[DoCa96] J. Dongarra and H. Casanova, "Netsolve"
http://www.cs.utk.edu/netsolve/

[Dong94] J. Dongarra "HeNCE: Heterogeneous Network Computing Environment"
http://www.cs.utk.edu/netsolve/

[FoKe96a] Ian Foster and Carl Kesselman, "The Nexus Multithreaded Runtime System",
http://www.mcs.anl.gov/nexus/index.html

[FoKe96b] Ian Foster and Carl Kesselman, "Globus: Infrastructure for Internet Computations",
http://www.mcs.anl.gov/globus

[FoKe96c] Ian Foster and Carl Kesselman, "I-WAY Software Infrastructure and Programming Tools"
http://www.iway.org

[FFCRC95] G.C. Fox, W. Furmanski, M. Chen, C. Rebbi and J. Cowie, "WebWork: Integrated Programming Environment Tools for National and Grand Challenges", June 1995
http://www.npac.syr.edu/techreports/html/0700/abs-0715.html

[FoFu95] G.C. Fox and W. Furmanski, "Use of the National Information Infrastructure and High Performance Computers in Industry", July 1995
http://www.npac.syr.edu/techreports/html/0700/abs-0732.html

[FuFo96] W. Furmanski and G. Fox, "Prototyping a WebVM based WebFlow Environment for Distributed Computing and Visual Programming", March 1996
http://www.npac.syr.edu/projects/webspace/doc/webflow/dual96.html

[Furm94] W. Furmanski, "WebTools – a CGI-extended Personal Web Server", NPAC, Syracuse University,
http://king.syr.edu:2006/WebTools.html

[CFLH96] G. Cheng, G. Fox, T. Lin and T. Haupt, "A Computing Framework for Integrating Interactive Visualization in HPCC Applications", NPAC, Syracuse University
http://www.npac.syr.edu/users/gcheng/homepage/index.html

[Grim95] Andrew Grimshaw, "Legion: Worldwide Virtual Computer"
http://www.cs.virginia.edu/~legion

[Haupt96] T. Haupt, "Early experiments with WebFlow interface",
http://aga.npac.syr.edu:1950

[Hk96] A three dimensional HPF implementation of a Black Hole evolution code
http://www.npac.syr.edu/projects/bbh/more.html

[Kla94] Klasky, Ph.D. dissertation, University of Texas at Austin (1994).

[Kla95] Klasky, Choptuik, and Matzner, "Interpolators/Extrapolators for AMR Finite Difference codes" Technical Report
, http://godel.ph.utexas.edu/Public/Members/scott/interp_doc.ps.

[Lasz96] Gregor von Laszewski, "Visual Parallel Programming", an NPAC applet, 1996
http://www.npac.syr.edu/users/gregor/java/Frame/

[NASA] The Numerical Propulsion System Simulator
http://www.lerc.nasa.gov/WWW/NPSS/

[NCSA] NCSA's Habanero Collaborative Tools Library
http://www.ncsa.uiuc.edu/SDG/Software/Habanero/ToolsLibrary.html

[NPAC96] NPAC Team, "Web based HPCC at NPAC" – List of Documents and Demonstrations
http://www.npac.syr.edu/projects/webbasedhpcc.html

[oracle92]The ORACLE7 Server concepts Manual, Part Number 6693–70–1292, Oracle Corporation, 1992.

[Par95]Distrubted Adaptive Grid Hierarchies
http://wwwrel.ph.utexas.edu/Members/parashar/DAGH/dagh.html

[Rice95] C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu and W. Zwaenepoel, "TreadMarks: Shared Memory Computing on Networks of Workstations", RIce University, Houston, TX.

[Rich10] Richardson, Philos. Trans. R. Soc **210** 307 (1910).

[RWC96]AFinalzer Bestavros, Marina Chen, Mark Crovella, Abdelsalam Heddaya, Stan Sclaroff, James Cowie (Co-operating Systems Corporation). "Responsive Web Computing: Resource Management Protocols, Techniques, and Applications." Boston University technical report 96–008. March, 1996.

[Spaff95] E.H. Spafford, "COAST — Computer Operations, Audit, and Security Technology", Purdue University
http://www.cs.purdue.edu/coast/coast.html

[Tuner95] Jonathan Tuner, "Distributed Control of ATM Networks", Washington University
http://www.arl.wustl.edu/~jst

[Veli95] B. Velikandanathan, "Parallel Search using a mesh of CGI-extended Web servers", NPAC, Syracuse University

http://kayak.npac.syr.edu:2007/HPDC4/Demos/WebWork.html

[VH]The NPAC Visible Human Viewer
http://www.npac.syr.edu/projects/vishuman/VisibleHuman.html

[Wheeler96]DAGH Applications
http://wwwrel.ph.utexas.edu/Members/parashar/DAGH/dagh.html

[Webman96] Framework." Cooperating Systems Corporation Technical Report TR-049603, April 1996. Available at
http://www.cooperate.com/TR/tr049603.html

[Wisc96] D.H.J. Epema, M. Livny, R. van Dantzig, X. Evers and J. Pruyne, “A Worldwide Flock of Condors: Load Sharing among Workstation Clusters”, University of Wisconsin
http://www.cs.wisc.edu/condor