Infrastructure requirements for a large-scale, multi-site VLSI development project

by G. P. Rodgers
I. G. Bendrihem
T. J. Bucelot
B. D. Burchett
J. C. Collins

This paper describes the design infrastructure and environment that were established to support the multi-site design of the IBM POWER4 microprocessor. The Common Tools Environment was created to provide a consistent means for accessing design tools and initiating operating system variables from multiple sites in a site-independent manner. The AIX® operating system and the Common Tools Environment masked local, site-specific details of the design environment, allowing site-specific design practices, shared storage, and information system policies to be transparently maintained. The design data-management system, the importance of highly reliable wide- and local-area networks, and the establishment of automated network monitoring are discussed.

Introduction

The design of the POWER4 microprocessor described in detail in this issue was a large, multi-site effort. The logical, circuit, physical, and verification design teams were composed of several subteams spread across two IBM divisions and five IBM sites. The IBM divisions involved in the design effort were the Server Group in Austin, Texas, East Fishkill, New York, Poughkeepsie, New York, and Rochester, Minnesota, and the Research Division in Yorktown Heights, New York.

Before the formation of the POWER4 design team, each subteam at the various sites had developed its own set of practices, policies, methodologies, and requirements that define the design environment within which each person works. A design environment includes such elements as

- Operating system.
- Default design-tool locations.
- Shared and private data storage locations and access permissions.
- Workstation access policies.
- Batch-processing policies.
- Design-tool problem tracking and resolution.
- Design-tool license access and accounting.
- Design methodology requirements.
- Design-tool release and qualification policies.

0018-8646/02/$5.00 © 2002 IBM

Table 1 Locations and management strategies for project data by type.

Data type	*File system*	*Data-management strategy*
Design data	DFS	Directory structure
Tool and reference data	DFS and AFS	CTE
Shared scratch data	DFS	Spooling
Final chip assembly	NFS	Cadence library consolidation
Local workstation data	JFS	Site administration guidelines

Additionally, the information technology support department at each site had its own set of practices, policies, and requirements:

- Backup policies.
- Scheduled maintenance.
- Problem identification, tracking, and resolution.
- Networking availability and performance standards.
- Help desk.

Each of these aspects of the design environment interacts with the others. To form a multi-site design environment, a common set of elements must be carefully defined so as to produce an enabling work environment. It is very easy to create a cumbersome, restrictive, and even conflicting set of rules that must be followed and endured by the design community, especially when formerly independent sites are initially combined. To be efficient, all designers at all sites should see a common design environment. Some aspects of the implementation details may be different, but the environment should at least appear to be common from the users' point of view. The design environment can enable multi-site design if done correctly, or inhibit it if done poorly.

POWER4 was designed on IBM RS/6000* hardware, ranging from single-processor user workstations to collections of centrally managed multiprocessor batch systems in large computing farms, all running on the IBM AIX* operating system [1]. The design-tool set was a combination of internally developed and vendor-purchased tools. The design methodology dictated that the team would use the best tool available for any particular function.

The POWER4 team successfully adopted common design-environment components that worked across all sites. Although far from a single, corporate-wide environment, the POWER4 effort successfully incorporated the best practices and required policies from each site. Two different distributed-data file systems were used to store POWER4 data. Each site utilized the Distributed File System (DFS*) [2] to store the actual chip design, while one site (Yorktown) used the Andrew File System (AFS*) [3] to store tool data and act as the master tool repository. A major component of the POWER4 design environment was the Common Tools Environment, or CTE.[1] CTE is a schema that allows design tools to be accessed in a common way that is site-, project-, and user-independent. If a tool interface is developed in a CTE-compliant manner, it will operate in the same way at any site, accommodating local differences and hiding them from the designer. Another important component developed for multi-site design is the CTE shadow process. This process is a means of distributing tools and synchronizing tool updates from a master site to all other sites, so that each design subteam has access to the current (or production-level) code, as well as selectable access to beta-level code. As described later in this paper, the CTE shadow process is network-aware, and works successfully with different shared-storage systems. Much was learned about sharing large amounts of data, communication, monitoring, and design-environment problem resolution among several physically separated sites. The common design-environment elements developed for POWER4 continue to evolve and serve as a foundation for other multi-site microprocessor programs within IBM.

Distributed data management

A large microprocessor design project consumes vast amounts of both shared and private data. Two enterprise-wide file systems, DFS and AFS, were used to manage shared data. Both were necessary because of the differences in data-management strategies at each site. All project workstations were required to be able to access both of these file systems. For performance reasons, the AIX native local file system, the Journaled File System (JFS) [4], was often used as temporary storage during a tool run before tool results were copied back to DFS. Also for performance reasons, final chip assembly used the Network File System (NFS**) [5] to share data among a subset of the project workstations. UNIX** environment variables were used to make data references as transparent as possible. **Table 1** shows how data was

[1] I. G. Bendrihem, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, private communication, July 31, 1996.

characterized and where it was located. The first four types of data require sharing among workstations and among users. The following sections describe how each of the various data types was managed.

Design data management

The design data is the primary product of a microprocessor design project. It is important to have a consistent strategy for managing such data. The following objectives were considered in creating the data-management strategy:

- Shared read access for all members of the project.
- Shared update access to a component of the data for all members of that component design team, also called a *unit* design team.
- Distribution among geographic sites for reasonable performance.
- Synchronization of logic design, simulation, verification, and physical design.
- Auditing of data to be sure of its validity.
- Access control to provide security and prevent invalidation of audit results.
- Support of the integration of data through multiple levels of design hierarchy.
- Overlapping of unit design schedules and chip integration schedules.
- Practical physical-volume management constraints for shared volumes.

Since DFS has intersite access-control mechanisms, it was used for all design data management.

To facilitate the sharing and site distribution of data, we created a high-level directory tree that was replicated at each site. The *nodes* of the tree, containing the low-level directory structure, were either local to a specific site or a symbolic link to another site. An environment variable pointed to the root of the tree, which was local to each site. This allowed consistent and transparent access to the design data. When referencing project data that was located at another site, performance could be affected, especially when trying to update design data not located at the local site. The physical location of the data was chosen so that it was most quickly accessed by the subteam responsible for creating the data.

The high-level directory structure was organized by chip, project, unit, major division of work, and release. The low-level directory structure was organized by type. The entire structure can be described as follows:

$CTEPROJPATH/<chipid>/<unit>/<division>/<release>/<type>,

where CTEPROJPATH was an environment variable that contained the name of the directory in which the project resided for the particular site.

The <chipid> was also an environment variable that indicated the revision of the chip being referenced. The <unit> designated the particular unit or team that created a component or related set of components of the chip. Examples of units were the floating-point unit, the fixed-point unit, the standard cell library, or the processor core. Data within a unit had to be organized by one of three major parts of the design process; we called these <divisions>. The three fixed divisions were *logic*, *sim*, and *PD*. These corresponded to logic design, simulation and formal verification, and physical design. This organization was intended to facilitate the flow of data among the different <divisions>. For example, after the logic design was completed, it was passed to the simulation and verification teams, and then to the physical design (PD) team.

The design of a typical component was a multi-pass process. Function was usually added progressively over time. When a component reached a certain level of function, a <release> of that component was made. It was important to identify this release and use this identification among the three <divisions> of work. All of the data associated with a particular release of a particular <division> of a particular unit was organized under the release directory.

One could say that our release-data management was done with a brute-force directory organization. This resulted in replication of data that was unchanged between releases. This inefficiency was accepted because no data-management tool existed that could span the wide variety of data types and various tool-access methods. One advantage of this practice was that we could use DFS to write-lock an entire unit-release directory tree after some level of qualification, so that it could be reliably used in the next level of the chip hierarchy for integration and assembly. For example, all units would use a read-only copy of the standard cell and component libraries.

The data within a release directory was organized by <type>. The directory tree below the type varied depending on the particular tool or method used to access the data. For example, physical design data such as schematics and layout were managed with type="cds". The Cadence Design Framework II** (dfII) [6] managed all of the type-"cds" data in Cadence libraries. DfII has its own revision control for iterations of a design within a release.

To ensure the consistency and quality of the design data, an auditing methodology was instituted. The audit checked such items as whether or not the data had been scanned for design-rule violations; whether time stamps were consistent; and what version of a particular tool had been used to operate on or generate the data. The

audit system assigned an overall grade associated with a particular release for a particular unit, and was organized around the release directory. When a unit design reached a certain level of audit, it was available for integration into the core or chip. Often, when a unit design team reached a particular audit milestone, they would move on to the next release of the design for that unit. This is how the overlap of schedules was accomplished.

Management of tool and reference data and associated replication technology

Most of the tool and reference data was read-only from the designers' perspective. A key requirement for these types of data was that they be consistent for all designers using the tools or libraries. As a result, replication methods were created so that tool and library access could always be performed locally to minimize network latency. Most of the design tools for POWER4 were managed in the CTE tool repository, as described later.

Reference libraries were copied or replicated to all sites when a library was released. We seldom allowed an update to a released library. Through configuration files, a unit release was associated with a particular level of a library. This would not typically change within the design period of that unit release. If a new reference library was available, the next unit release would use it.

Spool data

The tasks of simulation, verification, and synthesis were often done with background or batch jobs, submitted to a workload-management system. These batch jobs often required huge amounts of temporary data. One approach would have been to allocate large amounts of disk storage to each designer and let the designer manage this data. This would have resulted in large inefficiencies in disk utilization and large start-up times to acquire data for each designer. Often, tool designers were not cognizant of the space requirements of their particular tool. Furthermore, it is more important for the tool maintainer to focus on the technology being delivered with the tool than to have data-management expertise for each tool. This would also have resulted in a variety of data-management strategies. Some tools were conservative and deleted intermediate data and/or wrote over the previous job output, while others were extremely liberal and left the task of data management to the user.

To address this issue, we introduced the concept of *spooling* (actually borrowed from mainframe batch-management systems). Spooled data was automatically purged with a predictable policy. This allowed users to share spool disk space. It is important to have a documented policy and repeatable practice when dealing with spool data. This consistency guaranteed that designers had temporary space available for a sufficient period of time to complete post-analysis work. The Austin site had the largest spool, which consisted of 70 four-gigabyte volumes.

Spooling was implemented through the customization of application interfaces which were typically graphical user interfaces. The application user interface called a utility, *spoolsubmit*, for submitting a batch job. This utility created a new directory on a spool volume, copied necessary input files to the directory, and then called the appropriate batch job manager. The spoolsubmit utility could accommodate about 40 parameters, but required only one, the application command itself.

Spooling simplified the application-shell programming in three significant ways. First, since spooling allowed the application to use the current-directory paradigm, each job could run within a new directory. Users or application owners did not have to worry about writing over the results of a previous job. Second, spooling provided independence from the batch manager. An application-shell interface had to call only a single spool interface, spoolsubmit, and indicate with a parameter the type of batch system desired. Often this was selected by the user. Finally, because temporary job data was created in a consistent manner, post-job-analysis tools could be built on top of this standard directory organization. A user would typically find all job output in a personal $HOME/spool directory, organized by job type, part name, and job number. Sophisticated spool browsers made it easy to analyze, clean up, save, or promote the results of any batch job that used spooling. Promotion was required when the job generated data that was required for the design and promoted to the next level, or had to be saved for audit purposes.

Spooling also provided job-status monitoring, application-completion messages, and system diagnostics that could be used in the event of a failure. Since the spool was managed in a shared space, tool-support personnel could easily look at a job output to help diagnose a problem.

Spooling allowed for the creation of special-purpose job-control features that could be shared by all applications. For example, one option permitted the job directory to be moved to a local directory at the beginning of job execution, and then moved all of the results back to the spool at the end of the job. This had significant performance benefits for simulation and verification jobs that repeatedly opened and closed files or wrote large amounts of data.

Since the spool was a shared-update area, it was managed on the largest logical volumes available. Because spool data was temporary and not as critical as design data, these volumes did not require backup, or could be backed up with a lower priority than the design data.

Final chip assembly

The integration of all of the physical design data for the final chip assembly was the most data-intensive task for the project. The team responsible for chip assembly was known as the chip-integration team. Their function was to create the "release interface tape (RIT)," the final data set for release to manufacturing. The term RIT is historical, and IBM no longer uses tapes for this process. The RIT process could be quite long and often required processing times that extended through normal maintenance windows of the AFS and DFS file systems. Furthermore, larger-than-normal files and volumes were necessary to collect all of the data required for RIT. As a result, a special self-contained computing facility known as the *RIT room* was created. About a dozen IBM RS/6000 Model 270 workstations were consolidated on a single 100-megabit Ethernet network. In addition, three large NFS Version 3 data servers were created to export volumes to other machines in the RIT room on the same Ethernet LAN. The RIT room machines required DFS for copying release data from other unit teams to the NFS volumes. However, they were isolated from WAN, LAN, and site file-server outages because all of the necessary data and tools were copied to the file servers in the RIT room.

Management of local workstation client

It was not critical that workstation administration practices be identical at each site. All IBM sites that do VLSI development consistently use the AIX operating system for design. AIX ensures backward compatibility for applications compiled on older versions of AIX. This allows each site to maintain its own local IT practices. Consistent operation between sites resulted more from standard AIX practices than requirements from our multi-site project. No common network infrastructure or userid administration was required. The only mandatory design workstation requirements were 1) AIX Version 4.1 or later, 2) a DFS client, 3) an AFS client, and 4) a batch-processing client.

Three batch managers were available:

- LoadLeveler* [7], an IBM-developed manager, used at almost all IBM sites.
- Qman, an internally developed workload manager for logic simulation, used only in Austin.
- UNIX background for jobs that would run on the user's personal workstation.

Common design environment

Multi-site development efforts have a critical requirement that designers, regardless of where they are located, see a design environment with consistent elements. This is not a simple task. One example is the computer platform on which the design tools are run. Normally at IBM, each division, and often each site within a division, has its own IT organization in charge of creating and installing operating systems (OS) on their local computing resources. At the start of the project, various versions of AIX were present at each site. Even when two sites were running the same version of AIX, it was often the case that the sets of patches that each site had applied to their AIX image were different. In addition, AIX gives users unlimited flexibility in allowing them to customize their OS environment. Users can choose from a variety of OS shells (ksh, csh, bash, tcsh, sh, etc.) [8] and graphical interfaces (Common Desktop Environment, Motif**, etc.) [9]. At log-in time, each user can preset environment variables that can affect the way any tool can run. The challenge was to provide a design environment that would present a common interface to the design tools, regardless of any variations in OS level, environment, and setup.

Several design revisions, or RITs, are necessary to complete a multipass microprocessor design. The POWER4 project treated each RIT as a semi-independent project, with a tool and data set that were independent from those of the other RITs. Although tool sets were very similar, the versions of the tools used in each RIT could be, and often were, different. Most designers need to work on at least two RITs at the same time, or on one RIT for one project and another RIT for a follow-on, derivative product. The design environment had to allow designers to work simultaneously on multiple projects while logged in to their workstations. For each project/RIT combination, a simple script was provided that would start an X-Windows terminal session (aixterm), configured for a project/RIT combination. Within this window, it was guaranteed that the design environment was specific to one RIT, free from cross-project environment contamination (e.g. environment variable settings). Design tools started from the RIT-specific windows were guaranteed to belong to the proper tool set for that RIT, even when submitted to the batch pool.

The design environment used the facilities provided by the Common Tools Environment (CTE) to ensure that tools were accessed in a site-independent manner. CTE also provided the means to assign a set of tool versions to a specific RIT as well as to the tool repository.

Common tools environment

Multi-site projects have a site-independence requirement; that is, projects must continue to function regardless of whether a specific site is down or inaccessible; tool access must be free of site-specific dependencies, such as hard-coded paths to tools; and tools must be replicated across sites for good performance and accessibility. Design teams must see a common design environment and common tool interfaces, and must have seamless migration paths for the design environment from one project release to the next

and from project to project. The CTE enables common tool usage by defining an efficient, fully customizable design environment that is project-, site-, and technology-independent. It also provides a tool repository that ensures consistent access to design tools from site to site. There are advantages to keeping all tools under one directory structure. The CTE tool repository defines this structure. For instance, the CTE tool directory tree is shadowed, on a nightly basis, to several sites across the company. Multi-site projects can function only if their designers access the same code level, regardless of their location.

The POWER4 project used a combination of internally developed and vendor tools. The methodology team was free to choose the best tool for the job. This practice had the advantage of optimizing each step in the design process, but also presented many challenges such as data formatting, language, and tool compatibility. The design-tool framework application and the interface from which most tools were launched was Cadence's dfII.

CTE specifies how to access tools, either from Cadence's dfII environment via Cadence's SKILL** [10] programming language, or from the AIX shell. The specification allows for site, project, and user customizations. CTE also specifies how to install tools in the CTE repository. Tools must be self-contained; that is, the tool installation directory must contain everything the tool requires in order to run, or have dependencies only on other tools already installed in the repository. In addition, CTE also allows for tool versioning. CTE distinguishes between point tools and applications. Point tools are programs that exercise a particular aspect of the design methodology. Examples of point tools are Cadence [10] (netlist extraction, place and route), ACES[2] (simulation), and BooleDozer[2]* (synthesis). Applications are small programs or shell scripts that improve the designer's productivity. One example of an application is an interface to a point tool. Tool interfaces increase productivity by hiding the complexity of running a point tool. The CTE tool repository includes about 80 point tools and more than 100 applications. It also includes some basic technology data, such as design libraries, device models, and technology ground rules. It does not contain any project-specific design data, such as schematics or physical design layouts. Although the existence of a tool repository might imply that CTE is a design methodology, it is not. CTE does not specify which tools to use or how to use them. It enables any methodology and facilitates multi-site design collaboration.

Under the CTE repository, each tool is allocated its own AFS or DFS volume or set of volumes, depending on the size of a single version of the tool. As of this writing, 390 volumes have been allocated for more than 100 GB of storage. The master repository resides at the Watson AFS cell in Yorktown. For the POWER4 project, the repository was shadowed to the participating sites. The CTE shadow program uses an efficient, parallelized replication algorithm. Only files that have changed are copied to other sites. A batch job is generated for each volume that has changed since the last time the volume was shadowed. Typically, the shadow completes within 30 minutes per site. Tool maintainers can force a shadow at any time, from any site, via a client-server program.

Under CTE, a tool is accessed via an AIX shell, Perl [11], or a Tcl/Tk[3] [12], CTE-compliant "wrapper." The wrapper hides tool-requirement details, such as environment variables and paths, from users. CTE-compliant wrappers are sensitive to version control; that is, projects or users may switch to a different tool version on the fly without having to reinitialize their environment. The wrapper obtains tool location information either from a tool-specific CTE AIX environment variable or from the CTE parameter database. Default locations for tools are specified in a CTE global file, but they can be overridden at the site, project, and even user level. CTE-compliant wrappers are essential to guarantee site-independent access to tools.

Cadence's dfII environment was used to provide schematic and layout design entry and the means to launch tools that operated on the design data. Cadence's SKILL programming language provides an application program interface (API) for building forms or graphical user interfaces (GUIs), or tools interfaces and applications that can be launched from Cadence's framework. Although it is fairly easy to program using this API, GUIs built from it are hard to customize because they require forms to be instantiated on the screen before they can be tailored to project-specific needs. CTE provides an API that sits on top of dfII, which specifies how to build CTE-compliant tool interfaces and applications. The GUI and application defaults are stored in a database, providing easy customization at the site, project, and user level: No recoding is necessary. The CTE API is sensitive to version control, and it is independent of any changes Cadence might introduce to its own API from release to release. Currently, the CTE API contains more than 250 SKILL functions.

In addition to being a central component of the POWER4 design environment, CTE has enabled design tool convergence and environment commonality across IBM's major microprocessor efforts. It has become a framework standard for implementing tool convergence and commonality by allowing site, project, and user customizations while maintaining site and technology

[2] Internally developed IBM design tools.

[3] Copyright 1999, Brown University, Providence, RI. All rights reserved.

independence. Convergence has resulted in significant sharing of resources, adoption of common goals and methodologies, and the elimination of many previously duplicated efforts.

IT practices and support

From an IT support perspective, a number of challenges were encountered and problems overcome in establishing the multi-site design environment. The infrastructure was not built from the ground up for POWER4 specifically, but utilized network and file-system resources that were in existence and shared by other users. The challenge was to make the existing infrastructures at five physical locations work together as one, enhance them as necessary, and contain costs. An obvious problem could have been the successful integration of IT services at the various independently operated locations, but a tight coupling of IT services among the sites turned out not to be necessary. Code levels for the file systems and OS were independent, and multiple levels were employed. Individual networks were also deployed by location, with combinations of fiber-distributed data interface (FDDI), Ethernet, Token Ring, and ATM in use. However, planned outages were communicated among the locations and to the user community. Although the infrastructure services varied by location and were generally independent in execution, the user's data dependencies among sites had to be actively managed. Some special monitoring was performed to quickly detect intersite networking problems and aid in problem determination and resolution. As an example, probes were created in each of the POWER4 sites and were used to measure response time to the file systems at both the local and remote locations.

Additionally, one site also monitored tool and data availability, network response time, batch-processing facilities, and the proper functioning of the availability-monitoring applications. The state of the design environment was checked every three minutes, 24 hours a day. If a slowdown in communication or a loss of service occurred, the event was logged as a *down*, and a radio page was sent to a small group in the design team. This group would evaluate the situation and decide whether the problem was critical enough to take immediate action and coordinate with the IT team. This monitoring often prevented relatively small problems from becoming larger, work-stopping situations and kept the design team apprised of the situation during planned and unplanned outages. When the problem was resolved, an *up* event was logged, and another radio page was sent to the team. The logged events formed an availability database as viewed from the design team's perspective. Once a week, a summary report was generated and reviewed. In terms of absolute server availability, we managed to achieve an average of 99.98% availability.

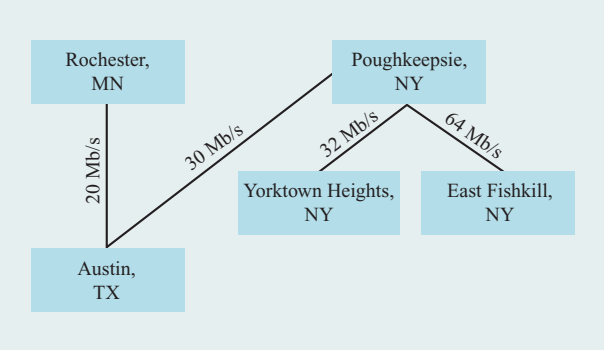

Figure 1

Wide area network capacity that supported the POWER4 multiple-site design project.

The standard IBM help desk, used by multiple IBM divisions, was used to record and track problems. As the POWER4 project progressed, each site developed additional services as needed. For example, Austin initiated a "walk-in" help desk to supplement the standard services. This gave engineers immediate, direct contact with support personnel. Yorktown instituted a code-word policy which was used to signify a high-priority POWER4 issue that would receive immediate, special attention. Through close cooperation between the design teams and the site IT departments, maintenance outages were scheduled around critical times in the design process. This communication and cooperation proved extremely valuable during critical phases of the design process.

At a file-system level, the multi-site design worked well. Standard networking response time between sites was typically 100 ms or less. In order for the network to be unnoticeable by a designer, a latency of approximately 50 ms to 75 ms would have been necessary. Much lower latencies (10 ms) were realized among some sites (Yorktown, Poughkeepsie, and East Fishkill). Although 100-ms latencies were sometimes inhibiting, they were not insurmountable. The POWER4 team shared the existing intersite network with many other users in IBM. It was impossible to know how many other users were present at any one time. No attempt was made, nor was it necessary to try to manage or schedule all user activities. Occasionally performance was degraded by very large file transfers, but in general the capacities shown in **Figure 1** proved adequate to support the POWER4 effort as well as all other traffic among the various sites.

Project communication

Effective communication among subteams located at different sites was an obvious requirement for the POWER4 project. In addition to significant amounts of

travel between sites, many tools and procedures were used to aid in this communication. Regularly scheduled voice and video conference meetings were heavily used. It soon became evident that video conferencing required at least 384Kb/s ISDN links to be effective. Lotus SmartSuite** and Lotus Notes** databases provided many communication tools and single points of access for status reports and presentations for all team members. A secure, project-wide Web site was used to provide access to methodology and project documentation as well as design team contact information.

Virtual classrooms were created with a workstation application-sharing tool known as *XMX*;[3] XMX creates a virtual root window running a second server on which shared applications are run. This enables workstation windows to be shared across multiple displays and multiple sites. XMX proved extremely useful for multi-site classes as well as for designer-to-designer communication and problem debugging. Another workstation tool, known as *Issues*,[4] was used to document concerns such as design-tool problems or networking issues. Design team members could *open* an issue in a database, which would automatically notify support teams as well as all others who subscribed to the issue category. This application provided a means for public dialog and status tracking for many tool and network issues.

Conclusions

We have presented an overview of the infrastructure requirements and design environment that were used in the IBM multi-site POWER4 design effort. The adoption of a Common Tools Environment across the sites, as well as the commonality of the computing platform and the AIX operating system, allowed the project to encompass many local, site-specific IT and project design practices and policies. The Common Tools Environment hid local, site-specific paths from the user and created a consistent, portable work environment for all of the subteams. We kept productivity high by maintaining a reliable network with adequate bandwidth, closely monitoring network connectivity and file accessibility, and coordinating problem identification and resolution.

Careful planning for the location of distributed data and the need to replicate certain files at each site enabled us to reduce, though not eliminate, network dependencies. We will apply the lessons learned about distributed data management, network monitoring, site-independent, network-aware tool wrappers, and intersite communication and cooperation to future joint programs that take advantage of corporation-wide resources.

[3] Copyright 1999, Brown University, Providence, RI. All rights reserved.
[4] Internally developed IBM tool.

Acknowledgment

The authors wish to thank Susan Beck, Peter Dudley, David Lewis, Kelvin Lewis, Robert Morel, Jeff Price, and John Jackson for their help and support in creating and maintaining the POWER4 multi-site design environment.

References

1. *IBM Certification Study Guide pSeries AIX System Administration*, IBM Redbook Publication No. SG24-6191-00, November 26, 2001; see IBM Redbooks Publications, *http://www.redbooks.ibm.com/*.
2. *IBM DFS 3.1 and DFS 3.1 for Web Enabled Environments*, Software Announcement 200-047, March 14, 2000; see *http://www.transarc.ibm.com/Product*.
3. See *http://www.transarc.ibm.com/Product/EFS/index.html*.
4. *AIX Logical Volume Manager from A to Z: Introduction and Concepts*, IBM Redbook Publication No. SG24-5432-00, January 20, 2000; see IBM Redbooks Publications, *http://www.redbooks.ibm.com/*.
5. *IBM DFS 3.1 and DFS 3.1 for Web Enabled Environments*, Software Announcement 200-047, March 14, 2000; see *http://www.transarc.ibm.com/Product*.
6. See *http://www.cadence.com/datasheets/analog_design_environment.html*.
7. *Workload Management with LoadLeveler*, IBM Redbook Publication No. SG24-6038-00, November 29, 2001, ISBN 0738422096; see *http://www.redbooks.ibm.com*.
8. *The Waite Group's UNIX System V Primer Second Edition*, SAMS, Prentice-Hall Computer Publishing, Carmel, IN 46032, 1992, pp. 313–321.
9. *RS/6000 Graphics Handbook*, IBM Redbook Publication No. SG24-5130-00, March 30, 1999; see IBM Redbooks Publications, *http://www.redbooks.ibm.com/*.
10. *SKILL Language Reference Manual*, Publication No. 900-60024-0101, Cadence Design Systems, San Jose, CA 95134, 1994.
11. Larry Wall and Randal L. Schwartz, *Programming perl*, O'Reilly & Associates, Inc., Sebastopol, CA 95472, 1992.
12. Brent B. Welch, *Practical Programming in Tcl and Tk*, Prentice-Hall, Inc., Upper Saddle River, NJ 07458, 1997.

Received July 20, 2001; accepted for publication December 19, 2001

Gregory P. Rodgers *IBM Linux Technology Center, 18 Frew Close, Nicholls, ACT 2913, Australia (rodgersg@us.ibm.com).* Dr. Rodgers joined IBM in 1981 as an MVS systems and application programmer. As part of the IBM Resident Study Program, he received his Ph.D. in computer science from Pennsylvania State University in 1989. Dr. Rodgers was a computer-aided design (CAD) development programmer for the IBM electronic design automation organization, working on parallel circuit simulation programs. From 1996 to 1999, he led the development and operation of the CAD environment for the POWER4 microprocessor development team. In 2000, he led a team that ported Linux to the POWER4. Today he continues to work on PowerPC Linux as part of the IBM Linux Technology Center on assignment to the IBM laboratory in Canberra, Australia.

Isidore G. Bendrihem *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (igb@us.ibm.com).* Mr. Bendrihem is a Senior Engineer in the VLSI Design Systems Department at the IBM Thomas J. Watson Research Center. He received the B.S., M.S., and M.Phil. degrees in electrical engineering from Columbia University in 1984, 1986, and 1988, respectively. He joined the IBM Research Division in 1992, working on several VLSI design tool issues for microprocessor design. Mr. Bendrihem is the architect for the Common Tools Environment (CTE), currently used at all major microprocessor development efforts at IBM. Current areas of research interest include design environments for project-independent, multi-site VLSI development. Mr. Bendrihem received an IBM Outstanding Technical Achievement Award in 1997 and a Research Division Outstanding Contribution Award in 1998 for his work in the design of the G4 and G5 S/390 microprocessors.

Thomas J. Bucelot *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (bucelot@us.ibm.com).* Dr. Bucelot is a Research Staff Member and Manager of the Design Systems Group at the IBM Thomas J. Watson Research Center. He received his B.S. degree in physics education from the State University of New York, College at Cortland, in 1969 and his Ph.D. in low-temperature physics from the University of Virginia in 1979. He joined IBM Research in 1979 to work on the Josephson superconducting computer project. From 1983 to 1992 he worked with and managed a group doing research in device and process measurements for liquid nitrogen and room-temperature-optimized CMOS technologies. In 1992 he joined the VLSI Design Department. Dr. Bucelot has received IBM Outstanding Technical Achievement Awards for the design of liquid nitrogen probe stations and for his contributions to the design of the G4 microprocessor. His current interests include multi-site design infrastructure issues and VLSI design-tool development for high-performance, low-power microprocessors.

Barry D. Burchett *IBM Global Services, 11400 Burnet Road, Austin, Texas 78758 (burchett@us.ibm.com).* Mr. Burchett received his B.S. degree in mathematics from the University of Kentucky. He joined ARDIS, a joint venture of IBM and Motorola, in 1990 as a network analyst and became operations manager. He joined IBM in 1996 as a manager of SAP delivery for IBM internal and commercial customers for Global Services. In 1998, Mr. Burchett became Delivery Project Executive for IBM Austin infrastructure, inclusive of file systems. Today, he remains with IBM Global Services as the Global Project Executive to the IBM Storage Technology Division. He also is an IBM Certified Project Manager.

John C. Collins *IBM Corporation, 11400 Burnet Road, Austin, Texas 78758 (jcollin@us.ibm.com).* Mr. Collins joined IBM in East Fishkill, New York, in 1969 and until 1984 worked as an engineer and manager in the high-performance chip design area. From 1984 to 1989 he was a Senior Manager in the large-systems channel-switching area. Mr. Collins joined the Austin Development Laboratory in 1990 and worked as a Project Manager in RS/6000 Development and Manufacturing until 1996. He then worked as a Project Manager with the POWER3 and POWER4 chip design teams in the IBM Microprocessor Development organizations. Mr. Collins retired from IBM in 2000 and worked as a Project Manager/Consultant for Schlumberger in the Chip Test operations organization. He returned to IBM in 2001 to work as a Project Manager Consultant. He received a B.S. degree in electrical engineering from Virginia Polytechnic Institute (1969) and an M.E. degree in industrial administration from Union College (1980). He also completed a Master's Certificate in project management from George Washington University (1999) and was certified as a Project Management Professional in 1999 by the Project Management Institute. Mr. Collins also holds patents and publications related to the technology and system area.