Performance analysis on a CC-NUMA prototype

by D. R. Kaeli
L. L. Fong
R. C. Booth
K. C. Imming
J. P. Weigel

Cache-coherent nonuniform memory access (CC-NUMA) machines have been shown to be a promising paradigm for exploiting distributed execution. CC-NUMA systems can provide performance typically associated with parallel machines, without the high cost associated with parallel programming. This is because a single image of memory is provided on a CC-NUMA machine. Past research on CC-NUMA machines has focused on modifications to the memory hierarchy, interconnect topology, and memory consistency protocols, which are all areas critical to achieving scalable performance. The research described here expands this focus to issues associated with operating system structures which can increase system scalability. We describe a hardware/software prototyping study which investigates how changes to the operating system of a commercial IBM AS/400® system can provide scalable performance when running transaction processing workloads. The project described was a joint effort between researchers at the IBM Thomas J. Watson Research Center and a team from the AS/400 development laboratory in Rochester, Minnesota. This paper describes various aspects of the project, including changes made to the operating system to enable scalable performance, and the associated hardware and software performance tools developed to identify bottlenecks in the existing operating system structures.

1. Introduction

A major roadblock to realizing scalable performance on a shared-memory multiprocessor system is the amount of data transferred between processor caches (and processing nodes). Ideally we would like to schedule jobs on processors or partition our data effectively to minimize this interprocessor or internode bus traffic. If we can capture workload characteristics through hardware monitoring and relate them to internode bus traffic, we can make specific changes to improve upon various aspects of the operating system responsible for such functions as task scheduling and memory allocation.

This paper describes an AS/400*-based prototyping study aimed at improving multiprocessor scalability for coarse-grained parallel commercial (transaction-based) applications. The system structure considered here comprises four single-processor, distributed processing nodes. Each node contains its own physically distributed node memory. **Figure 1** shows the basic configuration of our hardware system. While our prototype system contains only four nodes, we depict a system containing additional nodes, since we are interested in studying higher degrees of scalability. Each local memory is fully accessible from all other nodes, so that the combined node memories appear to be a single, linear address space. The nodes are connected by buses or switched links. The operating system, although partially partitioned, presents an image

0018-8646/97/$5.00 © 1997 IBM

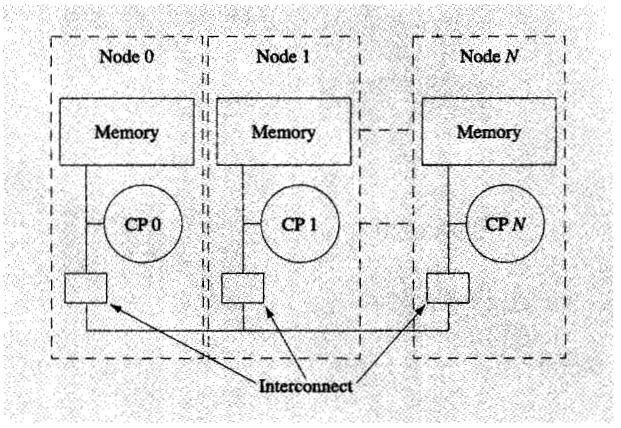

Figure 1

CC-NUMA AS/400 prototype system.

of a homogeneous multiprocessor system. Although similar systems have recently been explored [1, 2], the developments produced by these studies have been limited to memory management hardware and memory consistency policies. The workloads used in these studies have been limited to scientific and engineering applications [3, 4]. We focus instead on possible modifications to the operating system structures and evaluate these modifications while running commercial applications.

A prototype system has been constructed to assess scalability on our CC-NUMA system. The prototype hardware consists of four single-processor, bus-connected nodes, each with its own assigned portion of physical memory. This system was originally designed to function as a tightly coupled four-processor system, but by making substantial changes to the memory-allocation and task-scheduling portions of the operating system, we have been able to embed a distributed system structure within a tightly coupled AS/400 system.

The prototype software comprises a modified, object-based OS/400* commercial operating system [5] and two transaction-processing benchmarks. The prototype was modified to obtain complete, object-typed reference traces, as well as instrumentation samples. With every traced memory reference we are able to identify its source and destination on our system, as well as the source of the reference in the operating system or user application.

By using the traces and instrumentation data obtained on the prototype, changes were made in the task-dispatching and page-allocation algorithms of the operating system, and further traces were collected. The conclusion was that internode reference traffic could ultimately be reduced sufficiently to achieve near-linear scalability on a system comprising 16 processors running transaction-processing applications. While other CC-NUMA studies have demonstrated higher degrees of scalability, not one of these studies has addressed transaction-processing application environments.

This paper describes various aspects of the prototype project, including the system modifications and system monitoring tools, as well as proposed modifications to the final system and accompanying operating system software. Section 2 addresses a number of CC-NUMA design issues. Section 3 examines the set of tools we have developed for this research. Sections 4 and 5 provide results which demonstrate the value of these tools. Section 6 concludes with a summary of this work.

2. Distributed shared-memory issues

Flynn's well-known classification of computer systems divides machines into four categories: single-instruction single-data (SISD), single-instruction multiple-data (SIMD), multiple-instruction single-data (MISD), and multiple-instruction multiple-data (MIMD). This taxonomy is based on the amount of parallelism present in the instruction stream and the data stream [6]. A better model for differentiating the multiprocessor machines of today is to classify them on the basis of memory organization. The model structure chosen for our prototype system can be classified as a nonuniform memory access (NUMA) machine [7] or, more generally, a distributed shared-memory (DSM) machine. The specific model we discuss is called CC-NUMA (cache-coherent nonuniform memory access).

A CC-NUMA system locates a pool of local memory near each processor. Each processor additionally has access to the local memories of every other processor. The cost of accessing nonlocal memory (i.e., addresses that resolve to memory local to other processors) is typically much greater than the cost of accessing local memory. This cost is a combination of the greater latency (i.e., electrical distance) to nonlocal memory and the contention encountered in the processor/memory interconnect. Local caches are provided, and all caches are maintained to support a single image of coherent memory. This requires substantial communication between nodes when internode accesses take place.

The key design issue addressed by this prototype study is how to minimize the number of internode memory references generated by commercial processing workloads (TPC-A** [8, 9] and RAMP-C [10]). A major barrier to scalable performance on CC-NUMA machines is the large latency associated with internode references. To capture the cause of internode references requires precise measurement of the occurrence and source of all memory accesses. We need to identify which application or operating system component generated the memory

reference, and what data type (read/write) or object class was the target of the reference. To measure these reference parameters, high-resolution system monitoring tools were developed, and changes to the memory-allocation and task-dispatching algorithms of the OS/400 operating system were made to support these tools. The data obtained allow precise prediction of the gains associated with planned operating system modifications.

CC-NUMA systems are specifically designed with the assumption that most memory accesses will be resolved locally on the requesting node. To better understand the impact of memory access latencies, an analytical model of our proposed CC-NUMA system was developed. The model developed the worst-case scenario that could be tolerated while still achieving scalable performance. The model included submodels of the memory, local bus architecture, AS/400 processor, and scalable interconnect technology which would be available during the development time frame of this system. On the basis of this model, for our CC-NUMA system to achieve near-linear scalability (i.e., performance increasing linearly as additional processors are added), internode (between two bus-based node clusters) accesses must be limited to less than 5% of all accesses.[1]

To reach this level of reference locality, incremental system software changes were made. After each set of changes were implemented, additional measurements were made, and simulations were performed to judge the effect. The result is a series of system modifications required to achieve scalable performance. Next, we describe the tracing and monitoring tools used to identify the causes of internode traffic.

3. Typed traces on a CC-NUMA machine

Trace data have commonly been used to evaluate design trade-offs of proposed computer systems [11–13]. Traces have been used to evaluate the performance of memory hierarchies [14], branch predictors [15], and computer pipelines [16]. Many issues are associated with obtaining useful trace data:

1. *Tracing overhead:* How is the system perturbed in order to log a trace sample?
2. *Trace content:* What is the amount of data necessary to capture per trace sample?
3. *Trace length:* What is the appropriate number of records to capture for the proposed simulation study?
4. *Tracing speed:* What is the peak trace record production rate on the system under test?

A discussion of these issues can be found in [17]. While many of these problems apply to any tracing environment, tracing in a distributed multiprocessor environment introduces a number of additional concerns:

1. *Multiple execution streams*—Parallel capture and accurate time stamping become an issue here.
2. *Synchronization*—Any perturbation of the system can skew the interaction between multiple, competing tasks, generating a very different access stream.
3. *Timing dependencies*—Multiple tasks may be requesting service from a shared device (e.g., disk) which runs in real time.

A discussion of these concerns can be found in [18]. Many of these problems occur only when we attempt to use a trace taken from one multiprocessor system as input to a simulation of a second system. It is difficult to predict how a trace obtained on one CC-NUMA machine would execute on a second machine with a different hardware organization. The ordering of executions may differ greatly because of changes in queuing effects and shared data access. In this work we do not attempt to use the traces for this purpose (though we have used these traces to obtain cache miss information). Instead, we use the trace information to identify which elements in the operating system and user applications are responsible for internode accesses.

• *Real-time tracing and instrumentation*

To enable us to quantify the impact of our operating system changes and to predict the impact of proposed changes, precise tracing hardware and software tools were developed. In addition to allowing traces to be gathered in real time (i.e., without slowing down the system), we also provide the capability to profile the reference patterns by an object's type. **Figure 2** shows how the monitoring system interfaces with the prototype system. Again note that our prototype contains only four nodes, while we depict a system containing additional nodes.

Hooks are inserted through an additional AS/400 memory card which monitors all accesses to memory (the bus is effectively "snooped"). This fifth memory card is fully addressable from all processors in the system. A daughter card was designed that sits on top of the fifth memory card. The daughter card is addressed using a portion of the address range of the fifth memory card. This daughter card contains dual-ported SRAM which maintains information for all object types currently present in physical memory. This SRAM can be written to or read from by any processor. The hardware tools rely on software modifications to allow for the creation of object-typed information to be stored in the SRAM.

Our trace capture methodology takes advantage of the fact that on an AS/400 system, all user entities and nearly all system entities are stored in separate virtual memory

[1] IBM internal communication.

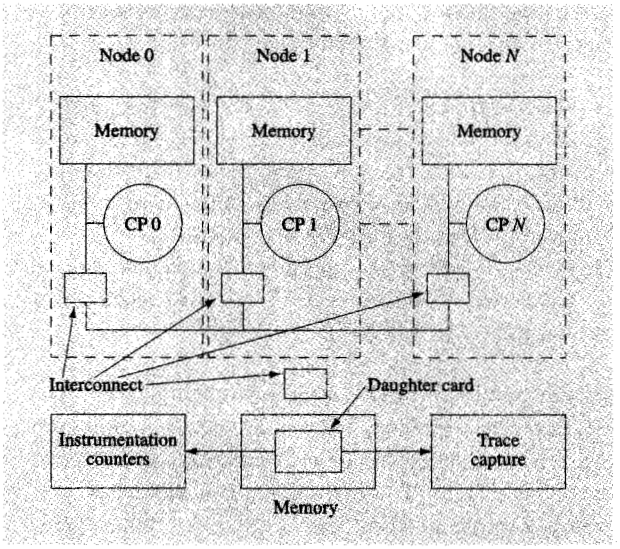

Figure 2

Tracing and instrumentation interfacing.

segments and are treated in an object-based fashion by the operating system. The virtual address space is partitioned into temporary and transient ranges. Each persistent segment has a permanent virtual address allocated from the permanent range. Private task objects are suballocated from special storage segments for efficient swapping. Further, a protected header in each segment contains or points to precise type information. By using these various page-type identifiers, we are able to determine page type at memory allocation or page-in time with minimal overhead.

This page-type information is stored (by the operating system) in a memory array (on the daughter card in the dual-ported SRAM) on our prototype. The memory array contains a byte of memory for every page in our physical address space. The operating system was modified to store page-type information in the memory array whenever a page is brought into memory or is reallocated. This stored information allows accesses to any particular page to be uniquely identified. The card monitors all memory bus accesses and indexes into the SRAM to identify what type of access is being performed. This information is read out and recorded, along with the address, requesting processor ID, read/write indication, and access length, and is captured by a real-time tracing system (see [19] for a description of this system). The real-time tracing system allows for capture of trace data without significantly perturbing the traced system (at a sustained rate of 25 megasamples per second). Any introduced overhead appears during an initial page allocation. The page type is dynamically deduced, and a store is issued to the page-type array. This results in a few extra instructions being issued on page allocations, which are typically infrequent events.

In addition to collecting a finite-length trace (64 million samples), we have the capability to count the various access types on our prototype system. We provide a bank of fast counters which effectively gives us an infinite time sample. The counter instrumentation (constructed out of programmable array logic) breaks down the accesses on the basis of requesting processor ID, target node memory, access type, and read/write. While object-type counts have proven to be useful in our work, they fail to capture the temporal patterns associated with memory addresses. For this reason we use both sources of data to guide our work.

- *Target workload*

Two different workloads are used in this work. RAMP-C is an IBM proprietary benchmark used to evaluate transaction-processing performance [10]. It consists of transactions of four complexity levels in accessing multiple data files. The files are in sequential, random, and indexed organizations, and the files are shared among transactions. TPC-A is an industry-standard benchmark which is based on the DebitCredit transaction-processing benchmark. An excellent discussion of the TPC benchmarks can be found in [8].

4. Object-typed performance data

To demonstrate the utility of trace and instrumentation information, we provide examples taken from our prototype project results which were used to identify the barriers to scalability on our system. We began by studying the access patterns on our unmodified system. From these data, we were able to plan a series of modifications to the software system which would limit the number of internode references.

Figure 3 plots the percentage of all accesses per object-type classification. These categories were chosen to be semantically meaningful and predictively useful. Each category represents as much as possible a collection of homogeneously used pages. Where necessary, references to individual data structures, such as single locks within a page, were isolated. Each category is represented by a unique bit pattern in the object-type memory. Further, each category represents a group of references which will be predictably affected by system modifications. The data were captured using the monitoring tools described previously.

- *Object types*

We take advantage of the fact that our AS/400 system makes use of its object-based nature. Object references on an AS/400 system can be identified with a fairly fine

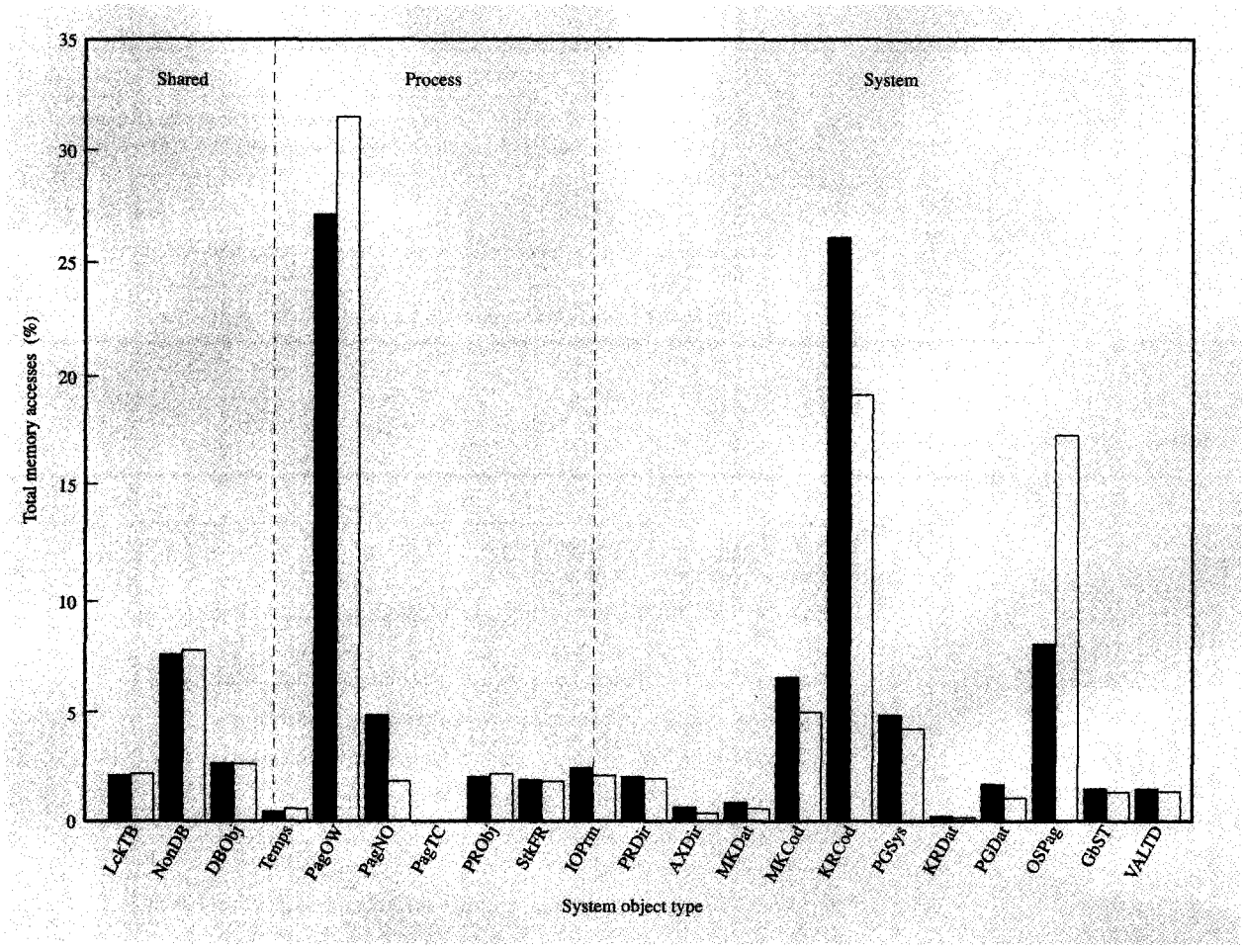

Figure 3

Access by object type: RAMP-C (dark bars) and TPC-A (light bars).

granularity. First, objects are grouped into one of three categories: 1) process (objects used by the operating system in direct support of a user's process); 2) operating system (objects identified as operating system data structures and codes, and not identified with any user processes); and 3) shared application (objects used by processes and system services). Process, operating system, and shared-application objects are described in **Tables 1**, **2**, and **3**, respectively.

In Figure 3, we see marked similarities between the two workloads. More than 50% of the memory accesses generated by both workloads are references to temporary storage for a process and nonpageable operating system code. The latter is not surprising, since the OS/400 database system is incorporated directly into the operating system.

This object-type breakdown is used to identify which object references in the system are responsible for internode references.

Figure 4 shows the breakdown of reads versus writes, and local versus nonlocal, for each object classification when running TPC-A. The *Reads* bar indicates the percentage of accesses which are reads (one hundred minus this percentage is the percentage of accesses which are writes). The *Locality* bar indicates the percentage of accesses which are resolved in the local memory (one hundred minus this percentage is the percentage of accesses resolved in nonlocal memory). We focus heavily on the locality results, since the success of our prototype project is heavily dependent on limiting the number of internode references.

Table 1 Process objects.

Type	*Description*
PagOW, PagNo, PagTC	Pag (process access group) is a container for process-specific temporary storage including invocation work areas, program automatic/static storage areas, and file/data path information
PRObj	Process control blocks
StkFR	Call stacks used in saving states for calls between user process and operating system functions
IOPrm	I/O request blocks containing parameters to be passed to I/O controllers

Table 2 Operating system objects.

Type	*Description*
PRDir	Primary storage directory which is embedded in the inverted page tables used in virtual address translation
AxDir	Auxiliary storage directories; used to control allocation and deallocation of disk space
MKDat	System microcode data including processor-related control information
MKCod	System microcodes for implementing complex system instructions
KRCod	Nonpageable operating system codes, excluding storage management functions
PGSys	Nonpageable codes for storage management functions
KRDat	Nonpageable operating control blocks, excluding those for storage management functions
PGDat	Nonpageable storage management controls, including storage management locks
OSPag	Pageable operating system codes
GbST	Transient storage (heaps) for operating system functions
VALTD	Nonpageable temporary storage objects, excluding StkFR and PRobj objects

Table 3 Shared-application objects.

Type	*Description*
LckTB	System-wide locking tables; infrastructure to support lock/unlock high-level instructions
NonDB	Permanent objects; data and codes of user application and system services which are not flagged as database objects
DBObj	Database objects, including indices
Temps	Temporary objects; data used by applications and system services

5. System modifications

The following describes the staged changes to the OS/400 operating system that had to be made to obtain near-linear speed-up on our proposed 16-processor system. These changes are selected and evaluated using the captured data on our prototype. Remember that our prototype consists of a four-processor, single-processor-per-node, CC-NUMA system, and that we are projecting results to a 16-processor, four-processor-per-node, CC-NUMA system. To support the validity of our results, we

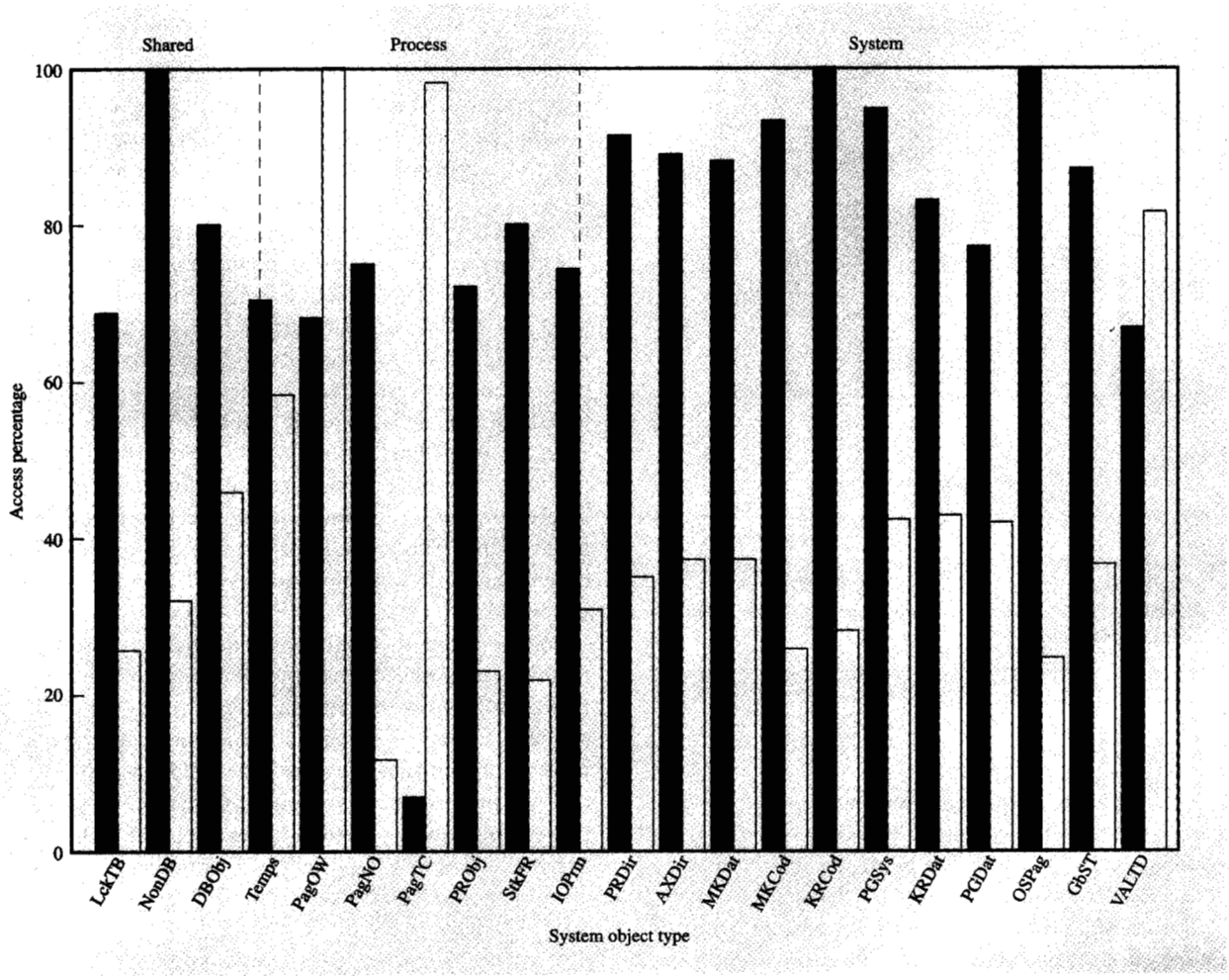

Figure 4

Access characteristics by object type (TPC-A): reads (dark bars); locality (light bars).

also captured measurements on a four-processor, tightly coupled multiprocessor system (i.e., the equivalent of a single node on a 16-processor CC-NUMA system).

Remember that our goal was to limit the number of internode references. We followed a set of incremental changes to our system to reach this goal. While it is beyond the scope of the prototyping project to implement all of these changes, we are able to estimate their effect on reference locality from the data capturing using the instrumentation tools available on our CC-NUMA machine.

In **Figure 5** we move through a series of stages, starting with the *Original* stage, which represents our original unmodified OS/400 operating system. *Stage 1* implements changes to task dispatching and page allocation. The task-dispatching changes made to our OS/400 system involved modifying dispatching structures to enforce node affinity [20]. The main idea here is to reschedule jobs on the node where they last ran (not necessarily the same processor) instead of migrating them to other nodes. This will limit the number of internode references due to job migration. Changes to the page allocation mechanism involved various strategies for placing newly allocated memory (on which node should the requested page be placed). Experimentation was done on allocating memory on the requesting or faulting task node, and basing this decision on the type of page that was being requested (e.g., the Pag objects, database data, operating system control areas). Since OS/400 is object-based, each page has a unique type that is identifiable upon allocation. Making these two major changes to the operating system, we were able to cut the number of internode references to 50%.

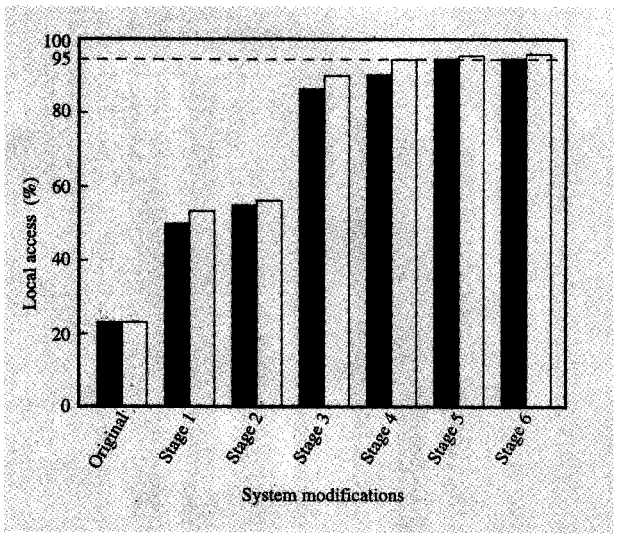

Figure 5

Locality for a four-node (16-processor) CC-NUMA: RAMP-C (dark bars) and TPC-A (light bars).

Stage 2 implements the use of local memory for the stack frames (StkFR), I/O parameter block areas (IOPrm), and process objects (PRObj). The internode references were reduced an additional 5%.

The remainder of the changes described are based on measurements produced using the customized instrumentation facilities. We specifically captured the counts of read and write data for both local and nonlocal references so that we were well positioned to make these projections accurately. The validity of the projected numbers is based on the following assumptions:

- Local clocking mechanisms do not introduce significant overhead to each node (they should actually simplify some operations).
- None of these modifications severely change the reference locality of accesses to other data types in the system.
- The changes in technology do not severely perturb the access patterns that we are currently seeing.
- Sufficient memory is added to the system to allow for the increase in code size generated by code replication and CISC-to-RISC code expansions.

The next stage implements R/ORep (read-only replication). In *Stage 3*, all read-only areas are replicated across each node. These areas include shared application and service codes (portions of NonDB), system microcodes (MKCod), and operating system codes (OSPag). This is a sizable improvement, as can be seen in Figure 5.

Stage 4 changes include modifications to the primary page table, hash tables (PRDir in the AS/400 uses an inverted page table [21]), servicing of I/O interrupts by the requesting processors (IOPrm allocated by a local node should be serviced by a local node), and replication of the hardware clock on each node (portions of MKDat).

Changes in *Stage 5* address modifications to the storage management locking structures to provide node-based locks in a hierarchical partitioning (PGDat, AxDir), and replicating server tasks on each node (e.g., hardware clock services and I/O service tasks). This pushes locality to the 95% local line, as indicated in Figure 5.

As the AS/400 platform moves to a PowerPC*-based processor, a natural code expansion will take place (caused by moving from a CISC architecture to a RISC architecture). This will further increase the locality of the code (estimated to be approximately 1%), and is accounted for in *Stage 6*.

We were able to effectively isolate the system software that was responsible for a large percentage of the internode memory references on our prototype system. Through a series of changes and additional instrumentation runs, we were able to design a system that will provide linear scalable performance, up to 16 processors. Very few systems today can boast such aggressive speed-up, especially when running transaction-processing workloads.

6. Conclusions

The results of this study have been used to influence future CC-NUMA designs within IBM. The insight provided by this study would not have been possible without the availability of such detailed performance data. We feel that the *typed* address trace approach used in this prototype study is a novel approach to providing software designers with insight into how their operating systems and system software execute on CC-NUMA machines.

Several important conclusions follow from the data obtained using these instrumentation tools. First is the quantitative observation that shared user objects (i.e., database objects and associated locks) play only a small role in the total references generated by transaction-processing workloads, and a correspondingly small role in CC-NUMA exploitation. We observed only approximately 5% of the total references to these categories. In marked contrast to engineering/scientific workloads, commercial transactions typically perform only simple data manipulations but require complex system support.

Second, system and user code, as well as process objects and structures, are responsible for the bulk of all references. We observed approximately 80% of all references to these categories. Thus, for these transaction-

based workloads, replication of read-only pages is absolutely crucial in reducing the number of nonlocal references. Equally critical, process pages must be placed local to the owning process at initial page allocation. The remaining references are made to a diverse set of system objects (i.e., tables, indexes, synchronization variables) which must be partitioned as much as possible.

Finally, to carry out correct page placement at page-in time, the OS must be structured to provide semantic hints about page content. This is a requirement for all but the most cursory page placement. In our case, important OS components are structured as objects stored in virtual memory segments with protected, self-typing headers. Further hints are provided through the virtual addresses assigned to persistent and temporary objects.

Acknowledgments

We would like to acknowledge the significant technical contributions of Dwight Renfrew and Valerie Nelson.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of the Transaction Processing Performance Council.

References

1. J. Kuskin, D. Ofelt, M. Heinrich, J. Heilein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, and J. Hennessy, "The Stanford FLASH Multiprocessor," *Proceedings of the 21st Annual Symposium on Computer Architecture*, Chicago, April 1993, pp. 302–313.
2. A. Agarwal, B. Bianchini, D. Chaiken, K. L. Johnson, D. Kranz, J. Kubiatowicz, B.-H. Lim, K. Mackenzie, and D. Yeung, "The MIT Alewife Machine: Architecture and Performance," *Proceedings of the 22nd Annual Symposium on Computer Architecture*, Santa Margherita, Italy, June 1995, pp. 2–13.
3. J. P. Singh, A. Gupta, and M. Levoy, "Parallel Visualization Algorithms: Performance and Architectural Implications," *IEEE Computer* **27**, No. 7, 45–55 (July 1994).
4. S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," *Proceedings of the 22nd Annual Symposium on Computer Architecture*, Santa Margherita, Italy, June 1995, pp. 24–36.
5. W. Berg, M. Cline, and M. Girou, "Lessons Learned from the OS/400 OO Project," *Commun. ACM* **38**, No. 10, 54–64 (October 1995).
6. M. J. Flynn, "Very High-Speed Computing Systems," *Proc. IEEE* **54**, No. 12, 1901–1909 (December 1966).
7. P. Stenstrom, T. Joe, and A. Gupta, "Comparative Performance Evaluation of Cache-Coherent NUMA and COMA Architectures," *Proceedings of the 19th Annual International Symposium on Computer Architecture*, Gold Coast, Australia, May 1992, pp. 80–91.
8. J. Gray, *The Benchmark Handbook*, 2nd Edition, Morgan Kaufmann, San Mateo, CA, 1993.
9. "TPC Benchmark, a Standard," Transaction Processing Performance Council (TPC), Los Altos, CA, ITOM International Co., 1989.
10. *IBM Application System/400 Commercial Performance*, IBM manual ZZ20-5871, June 1988; available through IBM branch offices.
11. S. J. Eggers, D. R. Keppel, E. J. Koldinger, and H. M. Levy, "Techniques for Efficient Tracing on a Shared-Memory Multiprocessor," *Proceedings of Sigmetrics '90*, May 1990, pp. 37–46.
12. Z. Cvetanovic and D. Bhandarkar, "Characterization of Alpha AXP Performance Using TP and SPEC Workloads," *Proceedings of the 21st Annual International Symposium on Computer Architecture*, Chicago, April 1994, pp. 60–70.
13. V. S. Iyengar, L. H. Trevillyan, and P. Bose, "Representative Traces for Processor Models with Infinite Caches," *Proceedings of HPCA-2*, San Jose, CA, February 1996, pp. 62–72.
14. A. J. Smith, "Cache Memories," *ACM Computing Surv.* **14**, No. 3, 473–530 (September 1982).
15. T. Yeh and Y. Patt, "A Comparison of Dynamic Branch Predictors That Use Two Levels of Branch History," *Proceedings of the 20th International Symposium on Computer Architecture*, May 1993, pp. 257–266.
16. M. S. Squillante, D. R. Kaeli, and H. Sinha, "Analytical Models of Workload Behavior and Pipeline Performance," *Proceedings of IEEE MASCOTTS*, January 1997, pp. 91–96.
17. D. R. Kaeli, "Issues in Trace-Driven Simulation," *Lecture Notes in Computer Science*, No. 729, *Performance Evaluation of Computer and Communication Systems*, L. Donatiello and R. Nelson, Eds., Springer-Verlag, 1993, pp. 224–244.
18. S. R. Goldschmidt and J. L. Hennessy, "The Accuracy of Trace-Driven Simulations of Multiprocessors," *Proceedings of Sigmetrics 1993*, Santa Clara, CA, May 1993, pp. 146–157.
19. D. R. Kaeli, O. LaMaire, W. White, P. Hennet, and W. Starke, "Real-Time Trace Generation," *Int. J. Computer Simulation Special Issue on Tracing Tools* **6**, No. 1, 53–68 (1996).
20. M. Devarakonda and A. Mukherjee, "Issues in Implementation of Cache-Affinity Scheduling," *Proceedings of the Winter 1992 USENIX Conference*, January 1992, pp. 345–357.
21. J. Feldman and C. Retter, *Computer Architecture: A Designer's Text Based on a Generic RISC*, McGraw-Hill Book Co., Inc., New York, 1994.

Received June 1, 1996; accepted for publication February 14, 1997

David R. Kaeli *Northeastern University, Boston, Massachusetts 02115 (kaeli@ece.neu.edu).* Dr. Kaeli received his B.S. in electrical engineering from Rutgers University, his M.S. in computer engineering from Syracuse University, and his Ph.D. in electrical engineering from Rutgers University. He is currently an assistant professor on the faculty of the Department of Electrical and Computer Engineering at Northeastern University. Prior to 1993, he spent twelve years at IBM, the last seven at the IBM Thomas J. Watson Research Center in Yorktown Heights, New York. Dr. Kaeli's research interests include computer architecture and organization, trace-driven simulation, I/O performance, and workload characterization. He is a member of the IEEE, IEEE Computer Society, ACM, SIGARCH, SIGMETRIC, SIGPLAN, Sigma Xi, and Eta Kappa Nu.

Liana L. Fong *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (fong@watson.ibm.com).* Ms. Fong is a Research Staff Member at the IBM Thomas J. Watson Research Center. Since she joined IBM in 1985, her research activities have been in the areas of operating system structures/algorithms, and performance evaluation for system architectures ranging from mainframes to workstations. She has worked on many aspects of the AS/400 system kernel and received an IBM Outstanding Innovation Award for her work on the OS/400 Scheduler. Ms. Fong received a B.S. degree in physics from the University of Washington and an M.S. degree in computer science from Northwestern University. She is a member of ACM and SIGOPS.

Richard C. Booth *IBM AS/400 Division, 3605 Highway 52 N., Rochester, Minnesota 55901 (rcbooth@rchland.ibm.com).* Mr. Booth is a Senior Engineer with the System Architecture group at the AS/400 Division Development Laboratory in Rochester, Minnesota. He received his electrical engineering degree from the University of Minnesota in 1974. He has held technical leadership and management positions in the development of IBM System/34, IBM System/36™, and IBM AS/400. In 1988, Mr. Booth completed a two-year technical sabbatical at the IBM Thomas J. Watson Research Center. During his time at Research, he implemented a message-passing multiprocessor system. Mr. Booth has received the IBM Outstanding Technical Achievement, Excellence, Invention Achievement, Author's Recognition, and Research Partners Awards; he has authored numerous technical papers and disclosures. His continuing interests are in the areas of scalable SMP and cluster architectures.

Kerry C. Imming *IBM AS/400 Division, 3605 Highway 52 N., Rochester, Minnesota 55901 (imming@rchland.ibm.com).* Mr. Imming is an Advisory Engineer in the SCU Development Department of the IBM AS/400 System Hardware Development organization in Rochester, Minnesota. His current responsibilities include development of the storage control unit for the future AS/400 PowerPC microprocessor. He has had assignments in processor design, memory card design, coordination of processor bring-up, and development of a firmly coupled AS/400 prototype system. Mr. Imming began his career with IBM as an associate engineer in S/36 Processor Development in 1984. He received an M.S. degree in electrical engineering from the University of Minnesota and a B.S. degree in electrical engineering from Iowa State University.

Joseph P. Weigel *IBM AS/400 Division, 3605 Highway 52 N., Rochester, Minnesota 55901 (jpweigel@rchland.ibm.com).* Mr. Weigel received his B.S. degree in computer science and electrical engineering from the University of Wisconsin at Madison in 1983. From 1984 to 1994 he worked on the various layers of the AS/400 microcode, predominantly in storage management. Mr. Weigel is currently a team leader in the Dev/2000 object-oriented tools group at IBM Rochester.