# Cluster-enabled OpenMP: An OpenMP compiler for the SCASH software distributed shared memory system

Mitsuhisa Sato[a], Hiroshi Harada[a],
Atsushi Hasegawa[b] and Yutaka Ishikawa[a]
[a]*Real World Computing Partnership, 1-6-1 Takezono, Tsukuba Mitsui-Blg. 16F, Tsukuba, Ibaraki 305-0032, Japan*
*E-mail:msato@trc.rwcp.or.jp*
[b]*NEC Informatec Systems, Ltd., Japan*

OpenMP is attracting wide-spread interest because of its easy-to-use parallel programming model for shared memory multiprocessors. We have implemented a "cluster-enabled" OpenMP compiler for a page-based software distributed shared memory system, SCASH, which works on a cluster of PCs. It allows OpenMP programs to run transparently in a distributed memory environment. The compiler transforms OpenMP programs into parallel programs using SCASH so that shared global variables are allocated at run time in the shared address space of SCASH. A set of directives is added to specify data mapping and loop scheduling method which schedules iterations onto threads associated with the data mapping. Our experimental results show that the data mapping may greatly impact on the performance of OpenMP programs in the software distributed shared memory system. The performance of some NAS parallel benchmark programs in OpenMP is improved by using our extended directives.

## 1. Introduction

In this paper, we present an implementation of a "cluster-enabled" OpenMP compiler for a page-based software distributed shared memory system called SCASH, on a cluster of PCs.

For programming distributed memory multiprocessors such as clusters of PC/WS and MPPs, message passing is usually used. A message passing system requires programmers to explicitly code the communication and makes writing parallel programs cumbersome.

OpenMP is attracting wide-spread interest because of its easy-to-use parallel programming model. While OpenMP is designed as a programming model for shared memory hardware, one way to support OpenMP in a distributed memory environment is to use a software distributed shared memory system (SDSM) as an underlying runtime system for OpenMP.

Our target SDSM is a page-based software distributed shared memory system, SCASH [2], which runs on a cluster of PCs connected by a high speed network such as Myrinet.

In most SDSMs, only part of the address space is shared. In SCASH, the address space allocated by a shared memory allocation primitive can be shared among the processors. Variables declared in the global scope are private in the processor. We call this memory model the "shmem memory model". Parallel programs using Unix "shmem" system calls use this memory model. In this model, all shared variables must be allocated at run-time at the beginning of execution. We have implemented the OpenMP compiler for our "shmem memory model" using the Omni OpenMP compiler system [7]. The compiler detects references to a shared data object, and rewrites them into the object re-allocated in the shared memory area.

The data mapping to processors is the key to achieving good performance on the SDSM. We have added a set of directives to specify data mapping and loop scheduling to give application-specific knowledge to the compiler. Using these extended directives, the programmer can exploit data locality by reducing the cost of consistency management.

Our contribution of this paper is to propose a technique to translate an OpenMP program for our "shmem memory model" of SCASH. We report performance and turning of OpenMP programs by our extended directives on our PC clusters. In the next section, we present an overview of the Omni OpenMP compiler system and SCASH, as background. Section 3 describes how to translate the OpenMP programs for the "shmem memory model", and Section 4 reports the

Scientific Programming 9 (2001) 123–130
ISSN 1058-9244 / $8.00 © 2001, IOS Press. All rights reserved

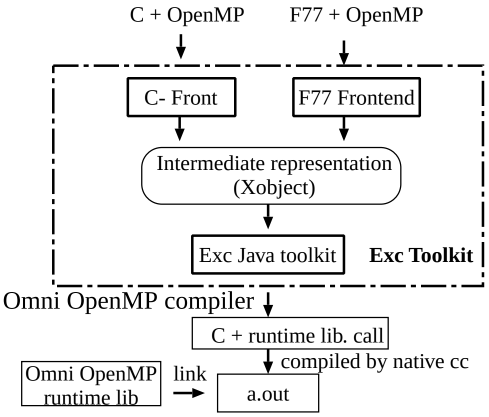


Fig. 1. An overview of the Omni OpenMP compiler.

performance obtained on our PC clusters with some benchmarks, including the NAS parallel benchmark suite. Section 5 presents our concluding remarks.

## 2. Background

### *2.1. The Omni OpenMP compiler system*

The Omni compiler is a translator which takes OpenMP programs as input to generate a multi-threaded C program with runtime library calls. The compiler system consists of the Omni Exc toolkit and various language front-ends, as shown in Fig. 1. C-front and F-front are front-end programs that parse C and Fortran source code into intermediate code, called Xobject code. Exc Java Toolkit is a Java class library that provides classes and methods to analyze and modify a program easily, with high level representation, and to unparse Xobject code into a C program. The representation of Xobject code is a kind of AST (Abstract Syntax Tree) with data type information, on which each node is a Java object that represents a syntactical element of the source code, and that can easily be transformed.

The Omni OpenMP compiler is implemented using the Omni Exc toolkit. The translation pass from an OpenMP program to the target multi-threaded code is written in Java using the Exc Java Toolkit. The generated program is compiled by the native back-end compiler linked with the runtime library. The OpenMP compiler is available for several SMP platforms, including Linux, Sun Solaris and SGI IRIX.

### *2.2. The software distributed shared memory system, SCASH*

SCASH is a page-based software distributed shared memory system using the PM low-latency and high bandwidth communication library [8] [1] for a Myrinet gigabit network and various memory management functions, such as memory protection, supported by an operating system kernel. The consistency of shared memory is maintained on a per-page basis. SCASH supports two page consistency protocols, *invalidate* and *update*. The *home* node of a page is a node that keeps the latest data of the page. In the invalidate protocol, the home node sends an invalidation message to nodes which share the page so that the page is invalidated on these nodes. The invalidate protocol is used as default.

SCASH is based on the Release Consistency (RC) memory model with the multiple writers protocol. The consistency of a shared memory area is maintained at a synchronization point called the “memory barrier synchronization” point. At this point, only modified parts are transferred to update pages. Explicit consistency management primitives are also supported for the lock operations.

## 3. The translation of OpenMP programs to SCASH

### *3.1. Transformation for the “shmem memory model”*

In SCASH, variables declared in the global scope are private in the processor. The shared address space must be allocated explicitly by the shared memory allocation primitive at run time. We call this memory model the “shmem memory model”.

In the OpenMP programming model, global variables are shared as the default. To compile an OpenMP program into the “shmem memory model” of SCASH, the compiler transforms code to allocate a global variable in shared address space at run time. The compiler transforms an OpenMP program by means of the following steps:

1. All declarations of global variables are converted into pointers which contain the address of the data in the shared address space.

[1] Currently, the PM communication library supports Gigabit and Fast Ethernet.

2. The compiler rewrites all references to the global variables to the indirect references through the corresponding pointers.
3. The compiler generates the global data initialization function for each compilation unit. This function allocates the objects in shared address space and stores these addresses in the corresponding indirect pointers.

For example, the following code:

```
double x; /* global variable
declaration */
double a[100]; /* global array
declaration */
...
a[10] = x;
```

is transformed into:

```
double *_G_x; /* indirect pointer
to ''x'' */
double *_G_a; /* indirect pointer
to ''a'' */
...
(_G_a)[10] = (*_G_x); /* reference
through the pointers */
```

The following initialization function `__G_DATA_INIT` is generated for the above code:

```
static int __G_DATA_INIT() {
 _shm_data_init(&_G_x,sizeof(double));
 _shm_data_init(&_G_a,sizeof(double)
 *100);
}
```

The run-time library function `_shm_data_init` specifies the size and the indirect pointer address for the shared object. The initialization function also contains the data mapping information if specified in the program. Figure 2 illustrates the code after this transformation.

The global data initialization function entry point is placed in the '`.ctors`' section[2] in order to be linked and called at the beginning of execution, before executing the "main" program. Actually, each initialization function only makes the table for the shared objects in each node. In the runtime initialization phase, the records of the shared objects in the table are summarized, and objects are allocated in the shared address space on the master processor (node 0). Then, the addresses of the allocated objects are broadcasted to initialize the indirect pointers in each node.

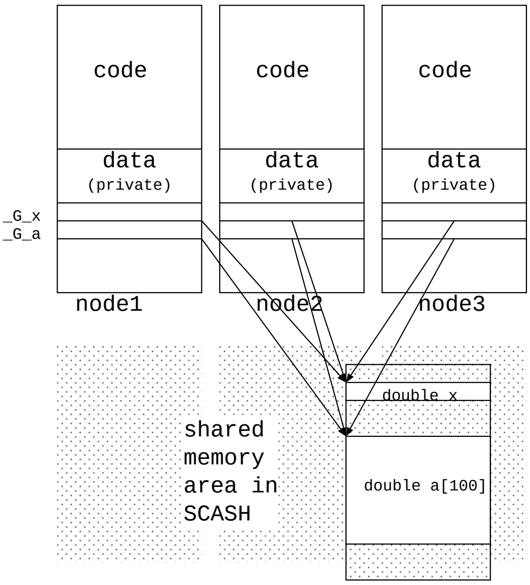


Fig. 2. Transformation for the "shmem memory model".

Note that if the shared memory was supported by hardware or/and operating systems as in SGI Origin 2000 the transformation described above would not be necessary. Our method is for a user-level distributed shared memory system, and does not need any modification of the operating system and the standard libraries. It just rewrites the references to the shared data objects and does not need complicated program analysis.

### *3.2. OpenMP directive translation and the runtime library*

The OpenMP directives are transformed into a set of runtime functions which use SCASH primitives to synchronize and communicate between processors.

To translate a sequential program annotated with parallel directives into a fork-join parallel program, the compiler encapsulates each parallel region into a separate function. The master node calls the runtime function to invoke the slave threads which execute this function in parallel. All threads in each node are created at the beginning of execution, and wait for the fork operation on slave nodes. At the fork, pointers to shared vari-

[2] The "`.ctors`" section means a "constructor" section which is originally used to place the "constructor" of C++ programming language. Each object file that defines an initialization function puts a word in the constructor section to point to that function. The linker accumulates all these words into one contiguous '`.ctor`' section.

ables with auto storage class are copied into a shared memory heap and passed to slaves.

No nested parallelism is supported.

In SCASH, the consistency of all shared memory areas is maintained at a barrier operation. This matches the OpenMP memory model. The lock and synchronization operations in OpenMP use the explicit consistency management primitives of SCASH on a specific object.

### 3.3. *The OpenMP extension for data mapping and loop scheduling*

In SDSMs, the home node allocation of pages affects the performance because the cost of consistency management is large compared to that of hardware NUMA systems. In SCASH, a reference to a page in a remote home node causes page transfer through the network. When the home node of a page is different from the current node, the modified memory must be computed and transfered at barrier points to update the page in remote nodes. SCASH can deliver high performance for an OpenMP program if the placement of data and computation is such that the data needed by each thread is local to the processor on which that thread is running. In OpenMP, a programmer can specify thread-parallel computation, but its memory model assumes a single uniform memory and provides no facilities for laying out data onto specific distinct memory space. And, no loop scheduling method is provided to schedule in a way that recognizes the data access made by that iteration.

We have extended the OpenMP with a set of directives to allow the programmer to specify the placement of data and computation on the shared address space. The data mapping directive specifies a mapping pattern of array objects in the address space. It is borrowed from High Performance Fortran(HPF). For example, the following directive specifies block mapping with the second dimension of a two-dimensional array A:

In Fortran:

```
 dimension A(100,200)
!$omn mapping(A(*,block))
```

In C:

```
double A[200][100];
#pragma omni mapping(A[block][*])
```

The asterisk (*) for the first dimension means that the elements in any given column should be mapped in the same node. The `block` keyword for the second dimension means that for any given row, the array elements are mapped on each node in large blocks of approximately equal size. As a result, the array is divided into contiguous groups of columns, with home nodes for each group assigned to the same node. The keyword `cyclic` ($n$) can be used to specify cyclic mapping. The alignment mapping directive is also provided to align data mapping of array to other arrays.

Since the consistency is maintained on a page-basis in SCASH, only page-granularity consistency is supported. If mapping granularity is finer than the size of the page, the mapping specification may not be effective. In contrast to HPF, each processor may have the entire copy of the array in the same shared address space. In this sense, this directive specifies “mapping” in the memory space, not “distribution” in HPF.

In addition to the data mapping directive, we have added a new loop scheduling clause, “affinity”, to schedule the iterations of a loop onto threads associated with the data mapping. For example, the iterations are assigned to the processor having the array element `a[i][*]` in the following code:

```
#pragma omp for schedule(affinity,
a[i][*])
 for(i = 1; i < 99; i++)
  for(j = 0; j < 200; j++)
   a[i][j] = ...;
```

Note that, in the current implementation, mapping and loop scheduling for only one of the dimensions can be specified because our current OpenMP compiler supports single level parallelism.

In C programs, a data is often allocated by the standard memory allocation function `malloc`. In SCASH, in order to allocate the data in a shared memory space, the SCASH-specific memory allocation function `ompsm_galloc` must be used in stead of `malloc`. This function takes the arguments which specify the data mapping of the allocated address space. The programmer may allocate the data mapped into a particular processor, or the data with block mapping.

### 3.4. *Compatibility with SMP programs*

From the viewpoint of the programmer, our implementation for the SDSM is almost compatible with one for the hardware SMP with the following few exceptions:

- In a cluster environment, I/O operations are performed independently in each node. The file descriptor allocated in a node cannot be used in different nodes.

Table 1
Execution time (in seconds) and Speedup in pcc2 (Pentium Pro 200MHz, 256MB memory, Myrinet network, Linux)

| No. of nodes | seq | 2 | 4 | 8 | 16 | 32 |
|---|---|---|---|---|---|---|
| lap/BLK | 17.74 (1) | 14.30 (1.24) | 7.84 (2.26) | 4.69 (3.78) | 2.88 (6.16) | 1.79 (9.91) |
| lap/RR | 17.74 (1) | 49.39 (–) | 33.88 (–) | 20.15 (–) | 12.87 (1.38) | 10.15 (1.75) |
| cg/BLK | 83.79 (1) | 48.90 (1.73) | 29.49 (2.84) | 20.86 (4.02) | 18.08 (4.63) | 19.65 (4.26) |
| cg/RR | 83.79 (1) | 55.20 (1.52) | 33.88 (2.47) | 23.33 (3.59) | 18.83 (4.44) | 19.73 (4.24) |

– In C OpenMP programs, the variables declared in external libraries must be declared as *threadprivate*. Global variables without *threadprivate* are re-allocated in a shared address space by the compiler.
– A dynamically allocated heap by using standard `malloc` is not shared. Use the `ompsm_galloc` instead, as described in the previous section.
– The number of threads is given by the command line of the SCASH system run command `scrun`, not by the OpenMP environment variable.

## 4. Performance evaluation

### *4.1. Data mapping and scalability*

We take two kinds of benchmarks to examine the effect of data mapping and scalability. The benchmark “lap” is a simple Laplace equation solver with a 5-point stencil operation (1024*1024 and 50 iterations). The benchmark “cg” is the NAS parallel benchmark CG (version 1) of class A. Both programs are written in C OpenMP. Table 1 shows the preliminary performance on the RWC PC Cluster II, (pcc2: Pentium Pro 200 MHz, 256 MB memory, Myrinet network, Linux). In SCASH, the home nodes are assigned to the pages in a round-robin manner in the order of address as the default. The execution time using this default home node allocation is indicated with RR. The execution time with BLK shows the performance when array objects are mapped by block mapping on the largest dimension. As a result, the large shared objects are equally divided into successive blocks for nodes. No scheduling clause is specified for any loops in either program. The column “seq” indicates the execution time of the sequential program compiled without any OpenMP directives.

In “lap” benchmark, we found that the home node mapping gives great impact on the performance. In each iteration, all elements in the large array are updated. At the barrier operation at the end of each iteration, modified elements are transfered to update the home pages, resulting in a large amount of traffic in “RR”. In BLK, the default loop scheduling matches the block mapping of the arrays. Most elements are referenced by the processor that is the home node for the pages containing the elements. The performance scales up to 16 nodes in this case. If a different loop scheduling such as cyclic scheduling was specified to the loop, the result would be greatly different.

In “cg”, we found that the data mapping has less of an effect than in “lap”, and its performance does not scale on more than 8 nodes. The major computation in “cg” is a sparse matrix vector multiplication. The large matrix is a read-only object so that the copy of the object is re-used in each iteration. The vectors are referenced and updated according to the data of the matrix. Since the elements in the vector are randomly referenced in this benchmark, all-to-all communication is required at the end of the parallel loops. This limits the scalability in SCASH.

For OpenMP programs for SCASH, the compiler maps the array objects using block mapping when no particular mapping is specified. This is sometime useful because the block mapping may match default loop scheduling in our compiler, as seen in the results of “lap”.

It should be noted that the overhead caused by rewriting global variable references by indirect pointers is very small in these benchmarks.

### *4.2. Performance tuning using the data mapping directives*

We have parallelized some benchmarks in the NAS parallel benchmark suite, based on serial version 2.3, using OpenMP. For parallelization, we simply added OpenMP directives, and did not modify the original code. To examine the scalability of these programs on hardware-supported shared memory multiprocessors, Fig. 3 shows the performance of our OpenMP version of BT and SP on a COMPAQ ProLiant 6500 (Pentium II Xeon 450 MHz, 1 GB memory, 4 CPU SMP, Linux) and a Sun E450 (Ultra SPARC 300 MHz, 1 GB memory, 4 CPU SMP, Solaris 2.6) by using the SMP version of our compiler.

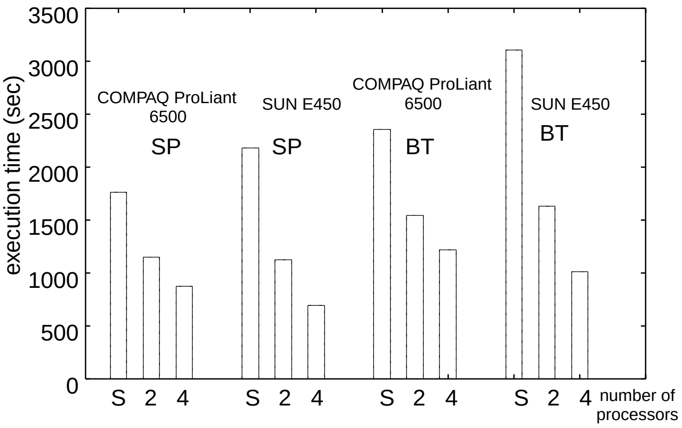


Fig. 3. Performance of hardware-supported shared memory multiprocessors.

Table 2
Execution time (in seconds) and Speedup of NAS parallel benchmarks in OpenMP on `compas` (Pentium II Xeon 450MHz, 1GB memory, Myrinet network, Linux)

| No. of nodes | seq | 2 | 4 | 8 |
|---|---|---|---|---|
| SP 2.3omp | 1807.7 | 1319.6 (1.37) | 821.3 (2.20) | 708.2 (2.55) |
| SP 2.3opt | – | 1235.8 (1.46) | 753.0 (2.40) | 462.7 (3.91) |
| SP PBN-3.0b | 1512.9 | 1441.7 (1.05) | 923.6 (1.64) | 610.8 (2.48) |
| BT 2.3omp | 2302.1 | 1413.0 (1.63) | 777.3 (2.96) | 491.0 (4.69) |
| BT 2.3opt | – | 1360.5 (1.69) | 754.8 (3.05) | 413.8 (5.56) |
| BT PBN-3.0b | 1456.2 | 1048.4 (1.39) | 621.6 (2.34) | 373.3 (3.90) |

Table 2 shows the performance of the NPB OpenMP benchmarks on our COMPaS Cluster (`compass`: Pentium II Xeon 450MHz, 1GB memory, Myrinet network, Linux). The size of all benchmarks is class A. The column "seq" indicates the execution time of the sequential program compiled without any OpenMP directives respectively.

We eliminated unnecessary barrier operations by adding "nowait" clauses at the end of some parallel loops to reduce unnecessary consistency traffic. If a variable can be referenced within its own processor locally, we made the variable "threadprivate" to reduce the amount of shared data. The rows indicated by "omp" indicate the performance using only the OpenMP standard directives.

PBN-3.0b (Programming Baseline for NPB) [6] is a new OpenMP benchmark suite of NAS parallel benchmarks released by the NASA Ames Lab. The programs in this suite are optimized from those of NPB 2.3. The rows indicated by "PBN-3.0b" show the performance of this benchmark on the SCASH.

We have tuned the performance of our OpenMP NPB benchmarks using data mapping directives and affinity loop scheduling. The rows indicated with "opt" show the optimized performance by our extended directives. Both benchmarks, BT and SP, solve multiple independent systems in three dimensional space which are stored in multi-dimensional arrays. For example, we specified block mapping with the second largest dimension for the four-dimensional array `rhs` in the benchmark SP as follows:

```
 dimension rhs(0:IMAX/2*2,
 0:JMAX/2*2, 0:KMAX/2*2, 5)
!$omn mapping(rhs(*,*,block,*))
```

One loop in the SP was scheduled as follows:

```
 do m = 1, 5
!$omp do schedule(affinity,
rhs(*,*,k,*))
  do k = 1, grid_points(3)-2
   do j = 1, grid_points(2)-2
    do i = 1, grid_points(1)-2
     u(i,j,k,m) = u(i,j,k,m)
      + rhs(i,j,k,m)
    end do
   end do
  end do
!$omp end do nowait
 end do
```

In this loop, the iterations for k are assigned to the processor which has the elements rhs(*,*,k,*).

The benchmark BT scales better than the benchmark SP because BT contains more computations than SP. Each sub-dimension of the array is accessed in three phases of x_solve, y_solve, and z_solve for each axis in the three-dimensional space. To parallelize these programs, the arrays are mapped with block mapping on the dimension corresponding to z. Although this mapping makes two routines, x_solve and y_solve, faster, the z_solve routine never scales because the data mapping does not match the access pattern of this routine. For example, most loops in x_solve ware parallelized at the most outer loop as shown in the above example. On the other hand, the loops in z_solve ware parallelized as follows:

```
 do k = 0, grid_points(3)-3
!$omp do
  do j = 1, grid_points(2)-2
   do i = 1, grid_points(1)-2
    k1 = k + 1
    ....
    rhs(i,j,k1,m) = lhs(i,j,k,m)
     + ...
    ...
   end do
  end do
!$omp do end
 end do
```

The most outer loop cannot be parallelized because the loop has the dependency between the iterations. The loops ware parallelized at the inner loop, resulting in mismatch between data mapping and iterations. To improve the performance of all routines, either remapping or re-ordering of data would be required to make the access pattern of each routine match the data mapping.

## 5. Related works

H. Lu et.al. [4] presents an OpenMP implementation for the TreadMarks [1] software DSM. Their compiler only supports a subset of OpenMP. Instead, they propose some modifications of the OpenMP standard to make OpenMP programs run easier and more efficiently. This approach may lose the portability of OpenMP programs. Our compiler supports a full set of OpenMP so that OpenMP compliant programs run on SDSMs without any modifications.

Hu [5] also presents an OpenMP for SMP clusters. They discuss performance on the modified TreadMarks software distributed shared memory system which uses POSIX threads with an SMP nodes.

The SGI OpenMP compiler also supports similar extensions to specify data mapping and affinity loop scheduling for their hardware-supported DSM system.

Bricak et al. [3] proposes similar extensions to OpenMP for NUMA machines in the COMPAQ OpenMP compiler. While we support only one-dimensional mapping with page granularity, they support multi-dimensional mapping and element-wise granularity by exchanging dimensions. The element-wise granularity would be a novel technique to exploit locality for our system.

## 6. Concluding remarks

We have presented an OpenMP compiler for a full set of OpenMP API on the software distributed shared memory system, SCASH, and examined its performance on our clusters. The compiler transforms OpenMP programs so that shared global variables are allocated at run time by SCASH primitives. Our implementation enables OpenMP programs to run transparently on the cluster environments with reasonable speedup, as shown in results of our experiment.

The page home mapping is the key to achieving good performance on the SDSM. We have added a set of directives to specify data mapping in a flexible way, which gives application-specific knowledge to the compiler. The loop scheduling used to exploit locality for data mapping can be used to tune the performance by reducing the cost of consistency management.

When the data access pattern does not match the data mapping, the performance degrades in the SDSM more seriously than in the hardware NUMA system. To improve the performance in programs which have different access patterns, remapping and re-ordering of data would be required.

## References

[1] C. Amza, A. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu and W. Zwaenepoel, Treadmarks: Shared memory computing on networks of workstation, *IEEE Computer* **29**(2) (Feb. 1996), 18–28.

[2] H. Harada, Y. Ishikawa, A. Hori, H. Tezuka, S. Sumimoto and T. Takahashi, Dynamic Home Node Reallocation on Software Distributed Shared Memory, In Proc. of HPC Asia 2000, Beijing, China, May 2000, pp. 158–163.

[3] J. Bircsak, P. Craig, R. Crowell, Z. Cvetanovic, J. Harris, C.A. Nelson and C.D. Offner. Extending OpenMP For NUMA Machines, In Proc. of Supercomputing 2000 (CD-ROM), Nov. 2000.

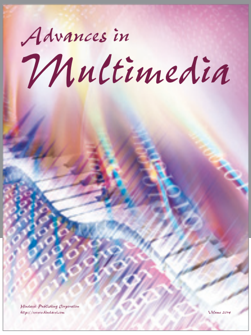
[4] H. Lu, Y.C. Hu and W. Zwaenepoel, OpenMP on Network of Workstations, In Proc. of Supercomputing ‘98 (CD-ROM), Nov. 1998.

[5] H. C. Hu and H. Lu, A.L. Cox and W. Zwaenepoel, OpenMP for Networks of SMPs, *Journal of Parallel and Distributed Computing* **60**(12) (Dec. 2000), 1512–1530.

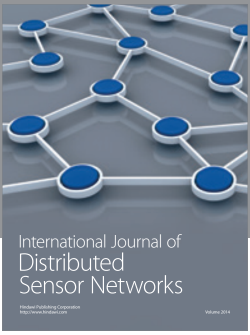
[6] H. Jin, M. Frumkin and J. Yan, The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance, NAS Technical Report NAS-99-011, Oct. 1999.

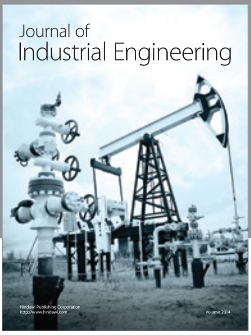
[7] M. Sato, S. Satoh, K. Kusano and Y. Tanaka, Design of OpenMP Compiler for an SMP Cluster, In Proc. of 1st European Workshop on OpenMP (EWOMP’99), Lund, Sweden, Sep. 1999, pp. 32–39.

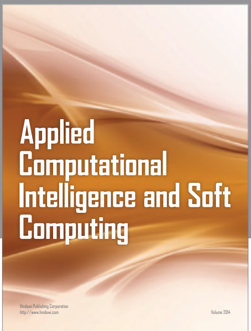
[8] H. Tezuka, A. Hori, Y. Ishikawa and M. Sato, PM: An Operating System Coordinated High Performance Communication Library, *Lecture Notes in Computer Science, High-Performance Computing and Networking*, 1997, pp. 708–717.

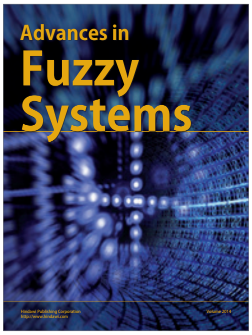

Advances in
Multimedia

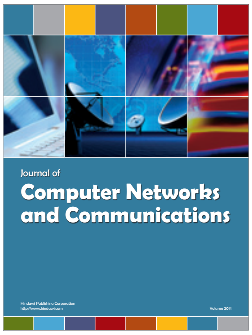
The Scientific
World Journal

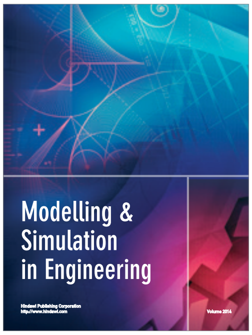

International Journal of
Distributed
Sensor Networks

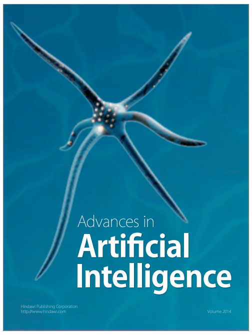

Journal of
Industrial Engineering

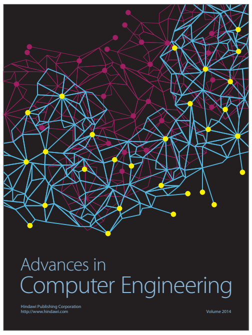

Applied
Computational
Intelligence and Soft
Computing


Advances in
Fuzzy
Systems

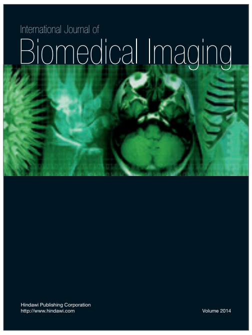

Journal of
Computer Networks
and Communications

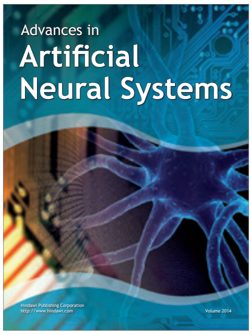

Modelling &
Simulation
in Engineering

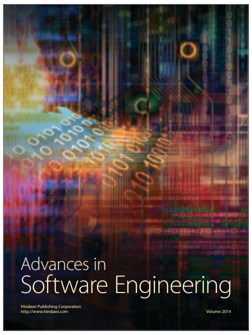

Advances in
Artificial
Intelligence


Advances in
Computer Engineering

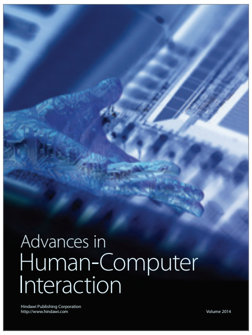
International Journal of
Computer Games
Technology

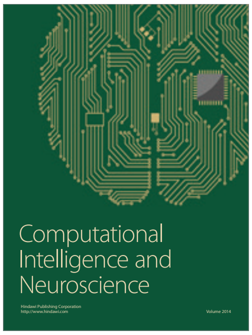

International Journal of
Biomedical Imaging


Advances in
Artificial
Neural Systems


Advances in
Software Engineering

Journal of
Robotics


Advances in
Human-Computer
Interaction


Computational
Intelligence and
Neuroscience

International Journal of
Reconfigurable
Computing

Journal of
Electrical and Computer
Engineering