OpenMP Experiences and Comparisons

Table of Contents
----

1. Site Background

The NAS facility has a long tradition with Cray vector platforms, but that technology may be nearing an end at NAS. The last C90, called vn, for Von Neumann, was a 16 CPU, 1 Gigaword memory computer, and was decommissioned 1/31/02.
An interim replacement, before a possible SV2 procurement, is the SV1ex, called bright, after Loren Bright. It has 32 CPUs, and 4 Gigawords memory, and was put into service 2/01/02.

There is also a long tradition of distributed shared memory machines from SGI, including currently the largest single-image Origin in existence, the 1024 CPU Chapman.
A partial list of currently installed Origins would include:

CPUS	NAME
1024	(Chapman) - 600 MHZ CPUs
512	(Lomax)
512	(Lomax3)
256	(Delilah)
128	(Hilbert)
62	(Steger)
32	(Turing)
30	(Hopper)
16	(Evelyn)
DAO	- 10 more systems
...	

Surprisingly, at the SGI Developers Conference, an SGI executive, in listing the largest systems, put GFDL, in Princeton, on top with over 1400 CPUs. Therefore, by his definition, total of all systems, NAS has a 3000 CPU Origin!

2. C90 --> SV1ex

Functionally, between the C90 and SV1ex, there are rather few differences, however, the following program, written by Dr. Johnny Chang, at NAS, illustrates how the machines differ by more than simply maintenance costs.

```
      program memintensive
! a memory access intensive program to check
!  timings between vn and bright
      parameter (niter = 10000, nmax = 1000000)
      dimension a(nmax), b(nmax)
      t0 = second()
      call random_number(a)
      t1 = second()
      do i = 1,niter
         call sub(nmax,a,b)
      enddo
      t2 = second()
      print *,'Time for random_number =',t1 - t0
      print *,'Time for memory copy   =',t2 - t1
      stop
      end
      subroutine sub(nmax,a,b)
      dimension a(nmax), b(nmax)
      b = a
      return
      end
```

This, and an analogous program, show quite clearly C90 and SV1ex memory and CPU differences.

For a memory-intensive code, the CPU time on SV1ex can be 4 times *slower* than on C90.

For a computation-intensive code, the CPU time on SV1ex can be 2.5 times *faster* than on C90.

However, when these results were broadcast, a representative of Cray pointed out that he felt they were unfair to the SV1, because they ignored the 32KW datacache which was available on the SV1ex, but not on the C90. And in fact, my tests suggest that if SV1 32KW datacache can be used, a 400 time improvement is possible.

Could this attention to the importance of 'multiple reuse of data' be a predecessor to 'cache orientation' on the Origins, at the beginning of code conversions from the Crays to the Origins?

3. SV1ex --> Origins

The SV1ex and Origin both 'parallel' machines, but...

The SV1ex has a vector orientation, and a flat memory layout. Also, SV1 cpus are not in general dedicated, so OpenMP thread commands are really 'requests', not commands.

The Origins have a distributed shared memory layout. Origin cpus, and their accompanying memory, are in general dedicated to the life of the job, and are called cpusets.

Whereas Cray options for parallelization, and memory distribution, tend to be 'inside' the compiler, using parameters, SGI has produced a long series of 'extensions', in especially two areas.

Parallel Thread Features

f90 compiler options

-pfa (from Kuck) -> -apo (separate license)

-LNO: - loop nest optimization

shmem - "MPI-like" shared memory library (from Cray)

```
 mp_shmem family - e.g. mp_shmem_get64 (...)

Memory Locality Features

 dplace - pre-execution NUMA memory placement tool

 pcf (Sequent, parallel computing forum, ANSI-X3H5 91-0023-B)

  C$DOACROSS, C$&, C$, C$MP_SCHEDTYPE
  C$CHUNK, C$COPYIN

  C$PAR BARRIER, C$PAR CRITICAL SECTION, C$PARALLEL DO
  C$PAR PDO, C$PAR SINGLE PROCESS, ...

 SGI extensions to PCF

 multiprocessing utility routines
  mp_block, mp_unblock, mp_setup, mp_create, mp_destroy,...
  mp_set_numthreads, mp_barrier, mp_slave_wait_for_work

 data distribution directives
  C$DISTRIBUTE, C$DISTRIBUTE_RESHAPE, C$DYNAMIC, ...

 Environment Variables
  _DSM_MIGRATION, _DSM_BARRIER, _DSM_PLACEMENT, ...
   (see man pe_environ)
```

So what does this have to do with OpenMP?

The SV1ex OpenMP was essentially a new implementation.

The OpenMP implementation on the Origins is generally based on PCF, DOACROSS technology and the SGI extensions.

So is OpenMP a complete upgrade of earlier SGI capabilities?

Not entirely. For example, there was the NESTED feature.

There was a particular form of NEST supported in PCF, which allowed you to exploit parallelism across iterations of a perfectly nested loop-nest, e.g.

```
c$doacross nest(i,j)
```

This is not in OpenMP, and is therefore, in a sense, a degradation on the Origins.

Parallelization of nested DO loops is defined in the OpenMP standard. However,..., not often implemented. I have been 'assured' this will be available in the coming months.

Is there an example of an SV1/Origin 'divergence' involving OpenMP?

For a line such as

```
!$OMP PARALLEL DO PRIVATE(iam)
!$OMP&  FIRSTPRIVATE(sum), LASTPRIVATE( sum)
```

The Origin f77 and f90 compilers produce

```
"prog.f", line 8: Error: FIRSTPRIVATE and LASTPRIVATE on same
 variable not yet implemented for PARALLEL DO
```

These lines compile correctly on the Cray.

However, the form

```
!$OMP PARALLEL PRIVATE(iam)
!$OMP DO FIRSTPRIVATE(sum), LASTPRIVATE( sum)
```

compiles on both platforms.

There are some tricks and guidelines which
may be helpful while testing OpenMP.

Use loc() to distinguish which data items
belong to which thread.
(But be careful if MPI is involved! Variables of the same thread
number spun off by different MPI processes may appear to have the
same location. They are distinct.)

Use sleep() to have time to view process activity.
(This is documented at 'man sleep 3C' on Cray, not in a 3F man page,
as on the Origins, and as one would expect.)

The order of control of the number of CPUs can be surprising.

Job control systems (nqs, pbs, lsf) may set defaults
or limit cpus. #PBS -l ncpus=4 (In PBS, for example)

On Cray, NCPUS precedes omp_num_threads.

On SGI, setenv OMP_NUM_THREADS 4 (csh environment variable)
or   call omp_set_num_threads(4) (internal OpenMP call)

In general, users would be well advised to always set for themselves
whatever CPU or thread values they require.

Process startup mechanisms vary between the two systems.

```
SV1ex - fork
        tfork (XMP task fork using ba,la)
Origins - fork
          sproc (for shared memory)
```

The Origins are threadsafe, the SV1ex is not.

SV1ex has f90 -a taskcommon.

SV1ex requires no compiler options for OpenMP
The Origins need -mp.

Origins produce a directory for rii_files,
used to facilitate data movement among nodes

SV1ex has 'real' nt and np mpirun options (nt would normally
be preferred).
On Origins everything runs -np.

4. MPI and OpenMP

Do MPI and OpenMP, when combined in the same program, behave more like 2 dogs or 2 cats? Dogs are social, work and play well together, complement each other. Cats are territorial, loners. They fight each other.

The idea is to create multilevel parallelism. The motivation can be better work distribution, or load leveling.

This idea is similar to a system called mlp, created by Jim Taft at NAS.

SGI recognized issues with data locality with these programs and has worked on an improved data placement scheme.

There is an SGI mod available in MPT 1.6 (May '02)
According to the SGI developer who is doing this enhancement:

The basic idea for this model is that the MPI processes are spread out to allow room for the OpenMP threads. The OpenMP threads for each MPI process are placed near the MPI parent. There is also an option to roundrobin the MPI process' data segment across the nodes that its threads are using. This has been found to help for higher thread/mpi process counts.

The model seems to benefit most applications where

1) the working data set does not reside in scache
2) more than 4 threads/MPI process

5. Conversion to OpenMP issues

What kind of problems occur in converting real codes?
In the world of OpenMP theory, almost all of the attention is on the parallelizing, variable handling, and scheduling of DO loops, and less frequently parallel regions. This is all swell, but how does a real world code differ from this portrait? Very likely, in the middle of that very natural DO loop you want to parallelize, there will be, not simply calls, but long call trees. These required a somewhat different focus than all of the theory referred to above. The book "Parallel Programming in OpenMP", which I highly recommend at the end of this paper, devotes around 8 pages out of more than 200 to dealing with these types of situations. I call the dichotomy horizontal/vertical code. The usual nomenclature is lexical/dynamic extent.

So what cautions are appropriate? I would propose an almost biblical admonition:

He who saves his code shall lose it.

SAVE, f77/90 -static can cause problems if the variables concerned should really be private.

If call trees are involved, TASKPRIVATE common blocks will likely be required.

In OpenMP 2.0 THREADPRIVATE may be applied to variables as well as COMMON blocks.

Converting codes to OpenMP, issues involving which variables need to be PRIVATE will probably need to be determined individually, not deduced from cpu=1 cases.

Or is the code parallelized from a past life?
That system may not match OpenMP completely!

Moreover one must watch out for issues of thread safety.
And remember that, since the order in which threads run varies, actions such as REDUCTION can have different values on separate runs. One must know the allowable variation in precision.

6. References

Parallel Programming in OpenMP (Chandra et al)

www.omp.org   is the primary site for OpenMP information.

www.nas.nasa.gov   for more information on activities ongoing at nas, and documentation on the systems there.