

## GAMESS - Application Execution Guidelines for vSMP Foundation Aggregated Virtual Machine

### Overview

GAMESS (General Atomic and Molecular Electronic Structure System) is an ab-initio quantum chemistry code. It provides many methods for computation of the properties of molecular systems using standard quantum chemical methods, many of them updated for parallel execution.

The source code of GAMESS benchmark is in FORTRAN, and a small amount of C source code. As such appropriate C and FORTRAN compilers are required. Intel compilers are recommended.

GAMESS uses a distributed data interface library known as DDI. DDI is built as part of the build and installation process of GAMESS. It is recommended to build DDI using MPICH2 tuned for vSMP Foundation.

### Sample building script using Intel compilers

```
---
#!/bin/bash

./opt/intel/Compiler/latest/bin/iccvars.sh intel64
./opt/intel/Compiler/latest/bin/ifortvars.sh intel64

export CC=icc
export CXX=icpc
export F77=ifort
export F90=ifort

export PATH=/opt/ScaleMP/mpich2/1.3.2/bin:$PATH
export HERE=`pwd`
export BASE=${HERE}/TI-06/app/games
export TAG=`date +%s`

rm -rf TI-06
tar xzf nsf_games.tar.gz || exit 1

cd $HERE
rm -rf games
tar xzf games-*.tgz || exit 1

cd $HERE/games
./config          # Sample answers are below
echo "setenv GMS_MKL_VERNO 11" >> install.info

echo cd $HERE/games/ddi
sed -i -e "s,set MAXCPUS = 8,set MAXCPUS = 1024," ddi/compddi
cd $HERE/games/ddi
./compddi >& compddi-$TAG.log || exit 1
```

```
echo cd $HERE/games compall
cd $HERE/games
./compall >& compall-$TAG.log || exit 1
```

```
echo cd $HERE/games lked
cd $HERE/games
rm -f games.00.x
./lked games 00 >& lked-$TAG.log || exit 1
```

----- Sample answers for config above -----

1. <enter>
  2. linux64
  3. <enter>
  4. ifort
  5. 11
  6. <enter>
  7. mkl
  8. /opt/intel/Compiler/latest/mkl
  9. <enter>
  10. x86\_64
  11. <enter>
  12. <enter>
  13. <enter>
  14. mpi
  15. mpich2
  16. /opt/ScaleMP/mpich2/1.3.2
- 

## Running GAMESS

### *DDI considerations*

DDI uses two communication mechanisms: MPI and shared memory. MPI is used between different hosts and shared memory is used within the same host. Since a vSMP system is practically a single system, out of the box DDI will regard it as a single very large shared memory system and will use shared memory for access to the distributed data (and will not use MPI at all). Since DDI is unaware of the large cache line size used by vSMP, the use of shared memory for communication by DDI causes a high volume of inefficient memory access activity. To solve this, a workaround was developed to cause DDI to use MPI for cross board communication. In effect the workaround causes DDI to think that the environment it is running on has multiple hosts. The workaround is implemented as a PRELOAD library. The code of this library is available in Appendix A. An environment variable named LOCAL\_NODE\_SIZE determines how many ranks are in each “pseudo host”.

DDI uses semaphores to control shared memory accesses. The subdivision of the MPI ranks into multiple “pseudo hosts” causes an increase in the number of semaphores used. As a result, on large vSMP systems, it might be necessary to increase the number of semaphores supported by the Linux kernel by performing the following command (or a similar one):

```
sudo echo 250 128000 32 4096 >/proc/sys/kernel/sem
```

## Environment variables

The following environment variables are used when running GAMESS with MPICH2 tuned for vSMP Foundation:

```
export PATH=/opt/ScaleMP/mpich2/1.3.2/bin:$PATH
export VSMP_PLACEMENT=PACKED
export VSMP_MEM_PIN=YES
export VSMP_VERBOSE=YES
```

Other environment variables:

```
export LOCAL_NODE_SIZE=<# of CPUs per board>
```

## Sample script

The following script sets up an environment to run a GAMESS benchmark with and then runs GAMESS on that benchmark.

```
#!/bin/bash

source intel
BASE=`pwd`
ulimit -s unlimited

export PATH=/opt/ScaleMP/mpich2/1.3.2/bin:$BASE/games:$PATH
export TMPDIR=<location of scratch folder>
export GPATH=$BASE/games
export MKL_SERIAL=YES

export LD_PRELOAD=$BASE/libddivsmp.so

mkdir -p $TMPDIR
rm -f $TMPDIR/*

# copy input files to current folder

export BENCHMARK="GAMESS"
export TAG=`date +%j-%H%M`
export RESULTSDIR=~ /Results/games/${TAG}
mkdir -p $RESULTSDIR

export VSMP_PLACEMENT=PACKED
export VSMP_VERBOSE=YES
export VSMP_MEM_PIN=YES
export LOCAL_NODE_SIZE=<# CPUs per board>

/sur/bin/time rungms <name-of-benchmark> 00 <#-of-ranks> \
    >& $RESULTSDIR/games.<name-of-benchmark>.<#-of-ranks>.log
```

## Appendix A

The following files are used in generation the workaround PRELOAD library libddivsmv.so

### Makefile

```
.PHONY: libddivsmv.so

libddivsmv.so: ddivsmv.c
    gcc -fPIC -shared -Wl,-soname,libddivsmv.so,-version-script,symver -o libddivsmv.so
ddivsmv.c
    gcc -g -O0 -o libddivsmv.test ddivsmv.c -lc -DCOMPILER_WITH_TEST_PROGRAM
clean:
    rm libddivsmv.so libddivsmv.test
```

### symver

```
GLIBC_2.2.5 { global: *; };
```

### ddivsmv.c

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

/*****/
#include <sched.h>
#include <string.h>
#include <sys/utsname.h>

#define DEFAULT_NODE_SIZE 8

int gethostname(char *name, size_t len)
{
    int i;
    int node_size = DEFAULT_NODE_SIZE;
    cpu_set_t aff;
    struct utsname utsbuf;
    size_t utslen;

    if (uname(&utsbuf))
        return -1;

    utslen = strlen(utsbuf.nodename);
    memcpy (name, utsbuf.nodename, len-1 < utslen ? len : utslen+1);

    if (utslen > len-1)
        return -1;

    if (len != 96) return 0;

    if (getenv("LOCAL_NODE_SIZE"))
        if (sscanf(getenv("LOCAL_NODE_SIZE"),"%d",&node_size) != 1)
            node_size = DEFAULT_NODE_SIZE;
    if (node_size <= 0)
        node_size = DEFAULT_NODE_SIZE;
```

```
    CPU_ZERO(&aff);
    sched_getaffinity(getpid(), sizeof(aff), &aff);
    for (i = 0; i < sizeof(aff)*8; i++)
        if (CPU_ISSET(i, &aff))
            break;
    if (i < sizeof(aff)*8 && strlen(name)+5 < len)
        sprintf(name+strlen(name), ":%d", i/node_size);

    return 0;
}

/*****/

void __attribute__((constructor))
my_init(void)
{
}

void __attribute__((destructor))
my_fini(void)
{
}

#ifdef COMPILE_WITH_TEST_PROGRAM
main()
{
    char buf[100];
    gethostname(buf, 100);
    printf("Test (ignored) HOSTNAME=%s\n", buf);
    gethostname(buf, 96);
    printf("Test (enabled) HOSTNAME=%s\n", buf);
}
#endif
```