HPCx
CAPABILITY COMPUTING

Performance of the turbulence code

Joachim Hein
HPCx Terascaling Team

May 8, 2003

Abstract

We report on the scaling of the turbulence code provided by the Departamento de Motopropulsion y Termofluidodinamica, E.T.S. Ingenieros Aeronauticos, Universidad Politecnica de Madrid, Spain on the HPCx system. For a problem size of $896^2 \times 897$ we observe good scaling and a good floating point performance.

1 Introduction

This report discusses the performance of a turbulence code written by members of the Departamento de Motopropulsion y Termofluidodinamica, E.T.S. Ingenieros Aeronauticos, Universidad Politecnica de Madrid, Spain on the HPCx system. In section 2 we briefly discuss the code and its parameters and section 3 gives an overview on the hardware and software used in this investigation. The performance of the individual parts of the code is discussed in section 4 and we close with a summary in section 5.

2 The Program

The program performs a turbulence simulation, using a third order Runge-Kutta solver. The program code is written in FORTRAN by Javier Jimenez, Roque Corral, Alfredo Pinelli, Juan C. del Alamo, Robert D. Moser, Paulo Zandonade. It is parallelised using MPI.
For this study we used $896^2 \times 897$ spectral modes. This requires a configuration file of 2.5 Gbyte of data which has to be read in the beginning of the program. In the appendix B we give the input files "ctes3D" and "hre.dat" as used for a 32 processor run of this study. For the above problem size, the number of processors has to be an exact divisor of 896. The authors of the program advised that due to dealiasing one direction becomes truncated to 595. To avoid load imbalance the number of processors should be close to a divisor of 595 as well.

3 HPCx system

3.1 Hardware

HPCx consists of 40 IBM p690 Regatta H frames. Each frame has 32 POWER4 processors with a clock of 1.3 GHz. This provides a peak performance of 6.6 Tflop/s and up to 3.2 Tflop/s sustained performance. The frames are connected via IBM's SP Colony switch. Per frame, these processors are grouped into 4 *multi chip modules* (MCM), each MCM has 8 processors. In order to increase the

communication band width of the system, the frames have been divided into 4 *logical partitions* (lpar), coinciding with the MCMs. Each lpar is operated as an 8-way SMP, running its own copy of the operating system AIX. Users are granted exclusive access to the lpars allocated. They are charged per lpar use, irrespective of the number of processors per lpar they use.
The processors inside an lpar share the same memory architecture, in particular there is only a single bus to main memory and also a single level 3 cache of 128 MB per lpar or MCM. The level 2 cache of 1440 kB is shared between all 2 processors and each processor has its own level 1 cache of 32 kB.

3.2 Software

For this test we used version *5.1* of the Operating system AIX. We used version *8.1* of the *XL Fortran for AIX* compiler. The files have been compiled with

```
mpxlf_r -c -O3 -qarch=pwr4 -q64
```

and linked with

```
mpxlf_r -O3  -q64 -bmaxdata:1200000000 -lessl
```

To achieve a low latency in the MPI system we set the environment variable `MP_EAGER_LIMIT=65536`.

4 Performance

The performance of the code has been measured for a range of lpar/processor combinations. In order to be able to make reasonable statements on the code's performance from short test runs, we timed the key sections of the code separately.

4.1 File IO

In the beginning the program has to read a restart file from disk. Towards the end of the program's execution and possibly at intermediate stages, the configuration has to be written back to disk. The present version of the code employs a master/worker model in these routines. Processor 0 reads/writes all the data and communicates them with the other processors involved. This part of the code does not scale at all. The times observed are essentially independent on the number of processors involved. For some runs we observed significantly longer times, which might be caused by other user's codes writing/reading at the same time. Typically we observed times around **90 seconds** for reading a configuration and around **120 seconds** for writing a full configuration. These times have to be compared to e.g. 30 seconds for a single iteration on 128 processors, see section 4.2. If the number of iterations between subsequent writes is large, these times should not result in a sizable overhead[1]. If on the other hand frequent writing of the configuration to disk is required for e.g. visualisation purposes, it should be investigated whether MPI-2 parallel file IO reduces the time required for writing. The present experience with parallel file IO inside the HPCx team suggests such a strategy to be beneficial to the performance [1].

4.2 Performance of the algorithm

In the test runs, we performed 5 Runge-Kutta iterations. Repeated runs of code showed only little variation in the iteration time. We did not observe any substantial difference between the time needed on the individual processors. We will quote the results as measured on node 0, the times for the other nodes differ by at most 2%, which is insignificant here. In figure 1 we report the time needed for the fastest of the five iterations of the run. Exact figures are quoted in appendix A. For the investigated

[1] 80 iterations between subsequent writes would result in a non-scaling overhead of about 5% for the writing

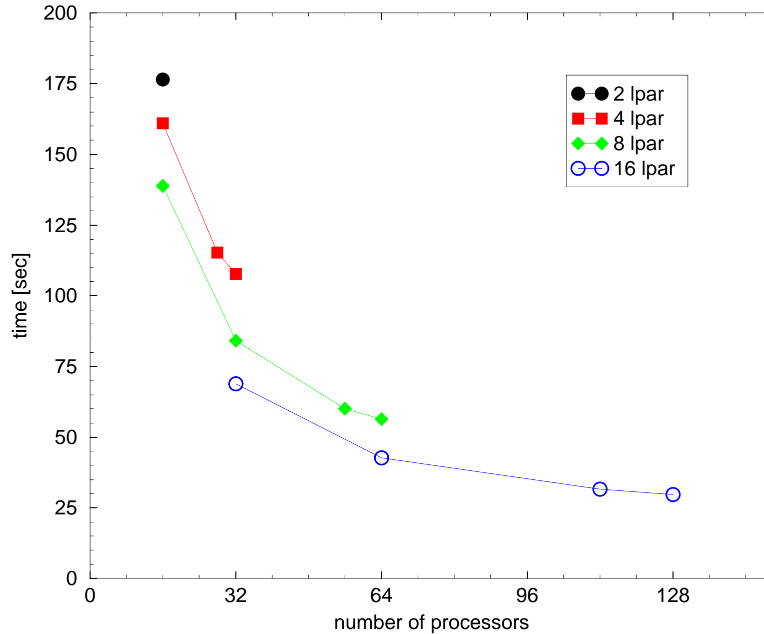

Figure 1: Time versus number of processors for a single 3rd order Runge-Kutta iteration. Results are shown for different numbers of lpars.

problem size it is not really feasible to use more than 128 processors, because of the reasons outline in section 2. Figure 1 shows that the best performance is achieved using 8 active processors per lpar for a fixed number of processors. Hence we would advise to use this value in production runs, since HPCx charges per lpars used.
In figure 2 we multiply these times by the number of processors involved in the calculation. This allows for the efficiencies to be read from the plot directly. To guide the eye, this time we connect points with an equal number of active processors. For a fixed number of processors, we note a drop in efficiency of about 40% when using 8 instead of 2 processors per lpar. This is most likely due to limited bandwidth from level 3 cache or main memory to the lpar. A performance degradation of this magnitude has been previously observed for other applications on HPCx [2].
When discussing the scaling we will start with the runs using 8 active processors per lpar. Figure 2 shows a good plateau for 32, 64 and 128 processors, that is for 4, 8 and 16 lpars. Between 32 and 128 processors the efficiency drops by only 10%. However, there is no clear picture with respect to the scaling from 16 to 32 processors. When using 8 active processors per lpar, the run on 16 processors is 20% more efficient than the one on 32 processors. On the other hand, when using only 2 or 4 active processors per lpar, there is no jump in efficiency between 16 and 32 processors. A proper investigation on the reasons for this behaviour is beyond the scope of this investigation.
Having observed good scaling of the code from 32 to 128 processor, one would like to know how well the code performs with respect to the floating point rate. We estimated this rate from the hardware performance counters. To read this counter for the Runge-Kutta iteration only, we instrumented the code using libhpm [3]. For processor 0 we get a rate of:

$$\text{single processor rate} = 390\ \text{Mflop/s}\,,$$

when running on 4 lpars with 8 active processors per lpar.

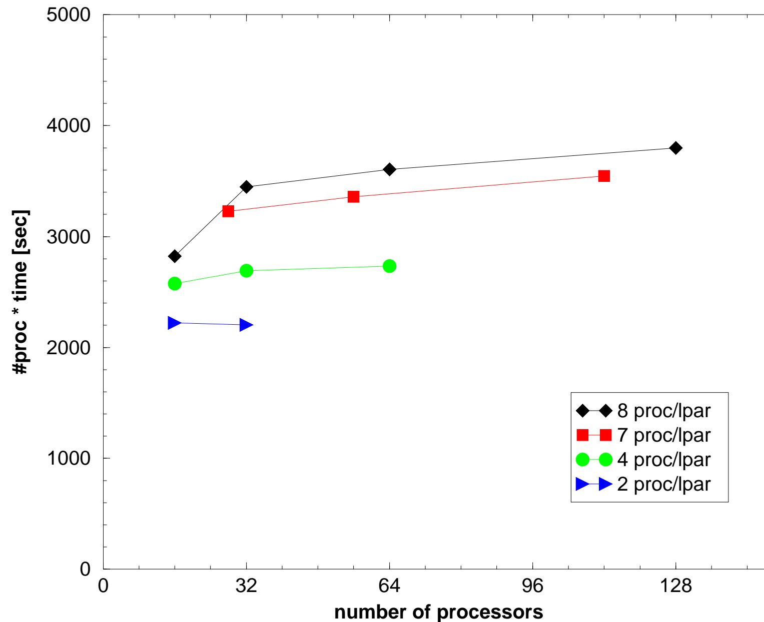

Figure 2: Time multiplied by processor number versus processor number. Results are shown for different number of active processors per lpar.

4.3 Further routines

The version of the code submitted for accessment contains two further section, which we timed. The first section calculates the Tchebychev spectra and the second performs measurements and writes them to disc. The input file 'hre.dat' controls how often this parts of the code are executed. To calculate the Tchebychev spectra it takes between 2% and 3% of the time needed for a Runge-Kutta iteration. The measurements routine, including the writing of the results to disc, takes about 1% of the time needed for an iteration. None of these routines results in a sizable overhead.

5 Summary

When using the supplied data set the code scales up to 128 processors and achieves a good floating point rate on HPCx. If the configurations are written infrequently, data IO will not result in a sizable overhead. If frequent writing of configuration files is required, rewriting the relevant routines using MPI-2 parallel file IO should improve the situation. We expect this code would make efficient use of HPCx.

References

[1] E. Breitmoser, private communication.

[2] J. Hein, "*A scalability study on 1280 processor p690 system*", talk given at ScicomP7, March 2003, Göttingen, Germany, `http://www.hpcx.ac.uk/research/hpc/`.

[3] For information on libhpm see:
`http://www.hpcx.ac.uk/support/documentation/UserGuide/HPCxuser/Tools.html`.

A Times for the Runge-Kutta algorithm

The following table list the measured times in seconds for a single Runge-Kutta iteration. These are the same results as shown in figure 1.

# proc	2 lpar	4 lpar	8 lpar	16 lpar
16	177	161	139	–
28	–	115	–	–
32	–	108	84	69
56	–	–	60	–
64	–	–	57	43
112	–	–	–	32
128	–	–	–	30

B Input files

B.1 File “ctes3D”

```
      integer numerop
      integer mgalx,mgalz,my
      integer mx,mz
      integer mgalx1,mgalz1
      integer mx1,my1,mz1
      integer mgx
      integer nz,nz1,nz2
      integer mgalzp,myp
      integer mzp,mgalz1p
      integer nxymax
      integer nspec
      integer jspecy

      parameter(numerop=32) !!!! NUMBER OF PROCESSORS

c       parameter(mgalx=512,mgalz=512,my=513)
c       parameter(mgalx=384,mgalz=384,my=385)
c       parameter(mgalx=1536,mgalz=1536,my=257)
c       parameter(mgalx=1280,mgalz=1280,my=257)
c       parameter(mgalx=1152,mgalz=768,my=257)
       parameter(mgalx=896,mgalz=896,my=897)
c       parameter(mgalx=512,mgalz=512,my=897)
c       parameter(mgalx=768,mgalz=512,my=193)
c       parameter(mgalx=128,mgalz=128,my=97)
c       parameter(mgalx=256,mgalz=256,my=97)
c       parameter(mgalx=384,mgalz=256,my=97)
c       parameter(mgalx=512,mgalz=512,my=97)
c        parameter(mgalx=768,mgalz=512,my=97)

      parameter(mx =2*(mgalx/3), mz = 2*(mgalz/3)-1)
      parameter(mgalx1=mgalx-1,mgalz1=mgalz-1)
      parameter(mx1=mx/2-1 ,my1=my-1, mz1=mz-1)
      parameter(mgx=mgalx/2)
      parameter(nz=(mz-1)/2,nz1=nz,nz2=mgalz-nz)
```

```
      parameter(mgalzp=mgalz/numerop+1,myp=my/numerop+1)
      parameter(mzp = mz/numerop+1,mgalz1p=mgalzp-1)
      parameter(nxymax=max(my*mzp,myp*mz))
C
C     ----------  planes for spectra ------------------
c      parameter(nspec=11)  !! re180, 23 planes
c       parameter(nspec=8)   !! re180, 17 planes
c        parameter(nspec=12) !! re550, 25 planes
c        parameter(nspec=13) !! re950, 27 planes
        parameter(nspec=21) !! re2200, 43 planes
c        parameter(nspec=17)  !! re750, 35 planes
       dimension jspecy(nspec)
c      data jspecy/ 8,11,14,17,22,27,33,36,40,43,46/       ! re180
c      data jspecy/ 8,11,14,17,22,27,33,40/                ! re180
c      data jspecy/ 7,10,12,15,19,23,26,29,37,44,49,61/    ! re550L
c        data jspecy/ 12,16,20,23,28,32,40,52,66,77,87,109/  ! re550H
c        data jspecy/ 14,19,23,27,32,37,46,59,80,99,115,130,163/ !re950H
        data jspecy/21,29,35,40,49,56,68,78,88,98,107,124,129,184,227,
     &             264,298,330,360,389,418/
c       data jspecy/ 8,11,14,17,21,26,29,33,35,41,50,58,65,
c     &              72,79,85,91 /                         ! re750L
c       data jspecy/ 1,3,6,9,12,15,18,21,24,27,30,33,36,39,
c     &             42,45,48/
```

B.2 File "hre.dat"

```
CC This file contains the input data to be used
CC in the 3-D DNS code for BIG CHANNEL at Kadesh
CC Structure of the present file:
CC 1) run size & no. of processors
CC 2) Parameters and utilities
CC 3) File names
CC NEW RUNS with parameters files changed!!!
CC-----------------------------------------------------------------------
CC 2)PARAMETERS used in the code
CC-----------------------------------------------------------------------
CC      Re       alp        bet          -u at
CC  Reynolds   x wave #   y wave        the walls
CC      *         *         *             *
CC   3.25d+03      1.0d0     2.0d0         0.
     5.2975e+04     2.        4.            0.53
CC
CC # steps   #step ims  #step hist # timestep     step v and p hist
CC    nstep    nimag     nhist      to throw away
CC                                stats and restart
CC      *         *         *         *                  *
       5      4        2      -1                   200
CC
CC  Time step   CFL   Start par: 0 gen. start; 1 restart ; 2 test
```

```
CC    Delt      CFL     istart
CC       *         *        *
     5.e-03   5.0e-01     2
CC
CC use Body  first     0/1/2        # steps
CC forces    output    0 nothing     between
CC 0 yes     file      1 start stat  statistics
CC 1 no      number #  2 read a file
CC       *         *        *        *
          0       24        1        1
CC
CC  first frame     last x      last z
CC    number        mode        mode
CC      *             *           *
        1            32          48
CC
CC  x modes pres  z modes pres
CC      *             *
       48           48
CC--------------------------------------------------------------------------
CC 3)FILE NAMES used in the code
CC--------------------------------------------------------------------------
CC file output max 50 char.
CC
Large_Data/outfile.dat
CC/scratch_tmp/juanc/peque2200
CC
CC  input file max 50 char.
CC
Large_Data/restart_file.dat
CC
CC  statistics file name max 50 char.
CC
Large_Data/stat_file_l4t8_timed.dat
CC
CC  movie file name max 50 char.
CC
Large_Data/movie_file_l4t8_timed.dat
```