

How to run jobs on the IBM Power4

Shuxia Zhang

Supercomputing Institute

e-mail: szhang@msi.umn.edu

help@msi.umn.edu

Tel: 612-624-8858 (direct)

612-626-0802(help)

Outline

Hardware and Configuration

Operations

Profiling and debugging tools

Queuing system

Hardware and Configuration

Four p690 (Regatta) SMP nodes:

- One 1.3 Ghz 32-processors, 64 GB of memory
- One 1.3 Ghz 24-processors, 24 GB of Memory
- One 1.7 Ghz 32-processors, 128 GB of memory
- One 1.7 Ghz 32-processors, 64 GB of memory

Thirteen p655 nodes:

- Each has 1.5 GHz 8-processors, 16 GB of memory.

Eleven fast p655 nodes.

- Each has 1.7 GHz 8-processors, 16 GB of memory.

One of these nodes is the interactive node.

Four 4-processor nodes that are used as file servers.

Network: IBM High Performance Switch (HPS)

Four scratch disks mounted on the Power4 system through
GPFS

/scratch1 : 800 GB

/scratch2 : 800 GB

/scratch3 800 GB

/scratch4 : 800 GB

Login Procedure

- You can only login to the interactive node: one 8-processor p655 node.
- This node is used for compiling and job submission.
- To run on the nodes of the Power4 system, you have to submit your job to the queue.
- To login:
`ssh -X -l username regatta.msi.umn.edu`

You only need to use `-l` if your login name is different than the login name you use on your own workstation.

Interactive Use

- There is only one node for interactive jobs.
- Interactive use should be restricted to compiling codes and code testing (both serial and parallel), and for submitting jobs to the queue.
- For parallel jobs, you may use up to 4 processors.
- Larger jobs and all production jobs must be submitted to the queues

How to compile - FORTRAN

Serial jobs:

```
xlf -O3 -qtune=pwr4 -qarch=pwr4 prog.f
```

```
xlf90 -O3 -qtune=pwr4 -qarch=pwr4 prog.f
```

```
xlf95 -O3 -qtune=pwr4 -qarch=pwr4 prog.f
```

MPI jobs:

```
mpxlf -O3 -qtune=pwr4 -qarch=pwr4 prog.f
```

OpenMP jobs:

```
xlf_r -qsmp -qtune=pwr4 -qarch=pwr4 prog.f
```

How to compile - C and C++?

Serial jobs:

```
xlc -O3 -qtune=pwr4 -qarch=pwr4 prog.c
```

```
xlCC -O3 -qtune=pwr4 -qarch=pwr4 prog.cpp
```

MPI jobs:

```
mpxlc -O3 -qtune=pwr4 -qarch=pwr4 prog.c
```

```
mpxlCC -O3 -qtune=pwr4 -qarch=pwr4 prog.cpp
```

OpenMP or SMP jobs:

```
xlc_r -O3 -qsmp -qtune=pwr4 -qarch=pwr4 prog.c
```

```
xlC_r -O3 -qsmp -qtune=pwr4 -qarch=pwr4 prog.cpp
```

How to run jobs interactively

Serial jobs:

```
./a.out < input > output
```

MPI jobs:

```
./a.out < input > output -nodes 1 -procs 2 -rmpool 1
```

man poe to get information about MPI ENV variables

OpenMP jobs:

```
setenv OMP_NUM_THREADS 1
```

```
./a.out < input > output
```

```
setenv OMP_NUM_THREADS 2
```

```
./a.out < input > output
```

Loadleveler - queuing system

You can specify a feature to request the type of node for your job:

Feature	NodeType	Number of Nodes	Specification
Regatta	1.3 GHz	2	#@ requirements = (Feature == "Regatta")
FastRegatta	1.7 GHz	2	#@ requirements = (Feature == "FastRegatta")
p655	1.5 GHz	13	#@ requirements = (Feature == "p655")
Fastp655	1.7 GHz	10	#@ requirements = (Feature == "Fastp655")

Wall Time limit

150 hours

24 hours

Nodes where job runs

All 32-processor Regatta nodes

20 p655 nodes

The 24-processor Regatta node

3 p655 nodes

Submit a serial batch job:

```
#!/bin/csh
#@ initialdir = /homes/r30/szhang/TESTS
#@ output = mm_serial.output
#@ error = mm_serial.error
#@ wall_clock_limit = 10:00
# requirements = (Feature == "FastRegatta")
#@ resources = ConsumableMemory(500)
#@ queue ./a.out
```

Note: No feature is specified in this example, so the job will run on any available processor on the system.

Submit a parallel MPI batch job

```
#!/bin/csh
#@ initialdir = /homes/r30/runesha/TESTS
#@ output = mpi_$(jobid).out
#@ error = mpi_$(jobid).err
#@ job_type = parallel
#@ wall_clock_limit = 20:00
#@ resources = ConsumableMemory(700)
#@ network.MPI = css0,shared,US
#@ blocking = unlimited
#@ total_tasks = 4
#@ requirements = (Feature == "Regatta")
#@ node_usage = shared #@ queue poe ./a.out
```

Note: This job will run any of the nodes with "Regatta" once 4 CPUs collectively available.

Submit an OpenMP batch job

```
#!/usr/bin/csh
#@ initialdir = /homes/r30/runesha/TESTS
#@ output = OpenMP_test.out
#@ error = OpenMp_test.err
#@ job_type = parallel
#@ wall_clock_limit = 30:00
#@ resources = ConsumableMemory(500)
#@ node = 1
#@ tasks_per_node = 4
#@ node_usage = shared
#@ queue
setenv OMP_NUM_THREADS 4
./a.out
```

Loadleveler commands

Submitting a job:

```
llsubmit test.ll
```

Monitoring job's status:

```
llq
```

```
llq -u username
```

```
showq
```

Canceling a job:

```
llcancel job_id
```

where `test.ll` is your LoadLeveler script and `job_id` is the job id that is returned when you submit a job to the queue.

Starting Totalview on MSI machines:

Compile the code with `-g` option added.

To start totalview:

```
% module load totalview
```

On a new process:

```
% totalview a.out -a <arguments>
```

On a core file:

```
% totalview a.out core
```

To attach to a running process:

```
% totalview
```

Tutorial hand-out on how to use totalview is available.

Xprofiler:

- Xprofiler is a GUI based performance profiling tool
- distributed as part of the IBM Parallel Environment for AIX.
- graphically identify which functions are the most CPU intensive in your code.
- It provides a graphical function call tree as well as a text profile pertaining to your code.
- Xprofiler can be used to profile sequential and parallel C, C++, Fortran 90, Fortran77.
- It could not provide information about I/O operation, though.

How to use Xprofiler?

Compiling and Linking your Program

For serial codes, do these

```
xlf -o executable -g -pg -O source.f
```

```
xlc -o executable -g -pg -O source.c
```

```
xlC -o executable -g -pg -O source.C
```

For MPI code, do these

```
mpxlf90 -o executable -g -pg -O source.f
```

```
mpicc -o executable -g -pg -O source.c
```

```
mpCC -o executable -g -pg -O source.C
```

Note: One can also include any compiler optimization options in the compiling. The "-pg" options must be included

How to use Xprofiler?

Generating the Profile

For serial jobs, type

```
executable < input > output
```

For MPI jobs,

```
poe executable < input > output -nodes 1 -procs 2 -rmpool 1
```

When job completes, a gmon.out files is created in the working dir.

For MPI job, files gmon.out.<*processor-id*> are created.

Notes:

Performance degradation is expected when running the the profiled version of your code.

How to use : Xprofiler?

Invoking xprofiler

To start xprofiler, you need to change to the directory where the *executable* and the *gmon.out.<processor-id>* files are stored, then type

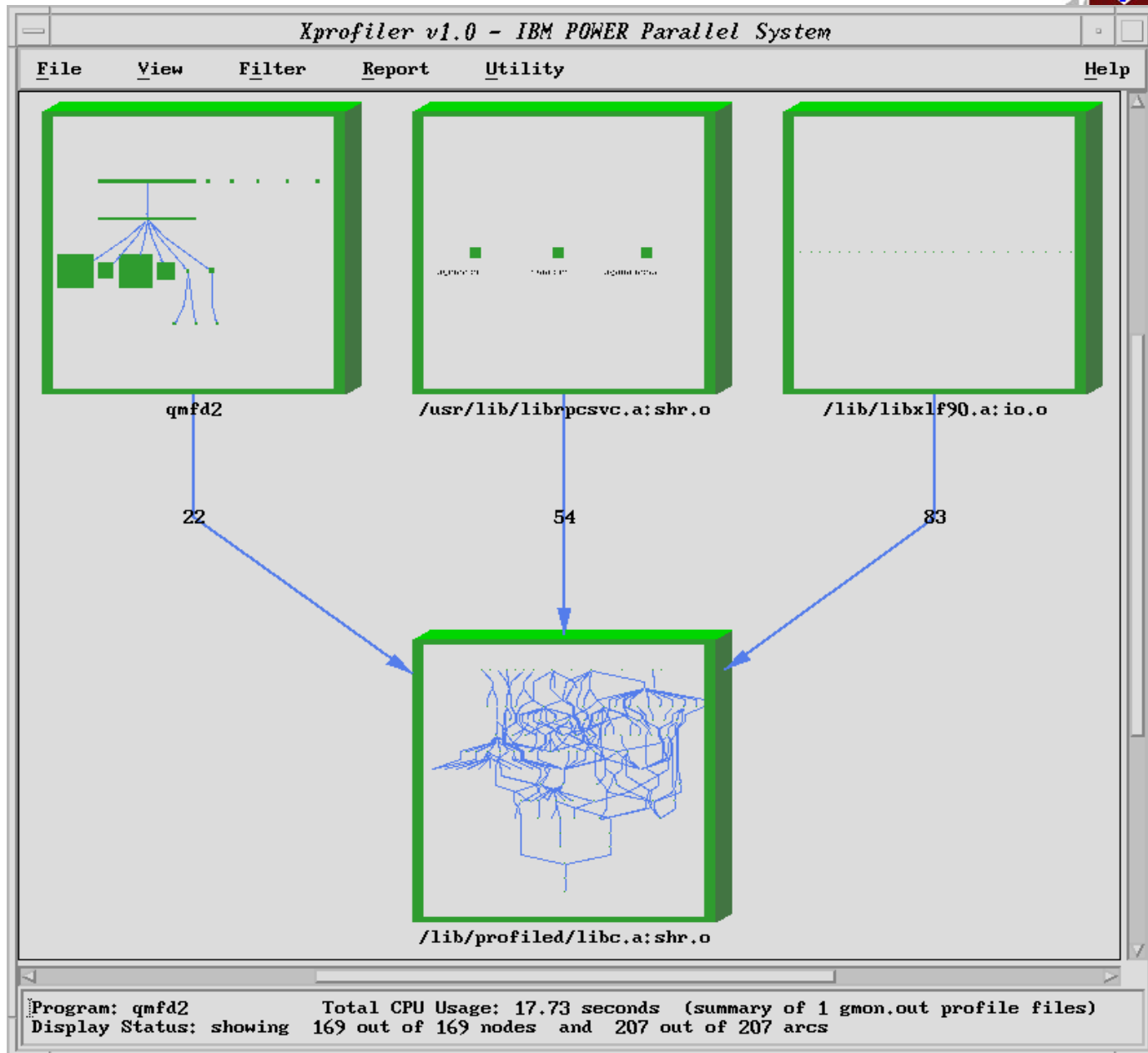
```
xprofiler executable gmon.out
```

Or

```
xprofiler executable [-s] gmon.out.<processor-id>
```

This will display the profiling information associated with this particular processor graphically

How to use Xprofiler?



Interpreting the Display

Xprofiler displays the profiled program in a single window. Green boxes represent functions and the calls between them are illustrated as blue arrows.

The size and shape of each function box indicates its CPU usage. The height of the function box represents the CPU time it spent executing the function itself, while the width of the function box is representative of the amount of CPU time it spent executing itself plus its descendant functions.

Each function box has a label stating the name of the function as well as the inclusive and exclusive CPU time values. The number of calls to a function are displayed on the blue arrow. Note it may be necessary to Zoom In i.e.

View -> Zoom In to see these labels.

By default, the display shows functions clustered by library. This may include shared libraries which contain system functions called by your application. To obtain a clear overview of the call tree for your executable only, use the **Filter -> Hide All Library Calls** option, followed by **Filter -> Uncluster Functions** .

Xprofiler Menus

Function Menu

A useful statistic is the *no. of ticks* per line of source code. This is useful to help locate the line(s) of source code where most CPU time is consumed.

Text Profile

The text profile pertaining to your application is available from the **Report** menu. The **Flat Profile**, **Call Graph Profile** and **Function Index** reports are identical to the output from the gprof profiler.

Library Statistics option provides timing information on the various libraries and external objects which are employed while the **Function Call Summary** reports statistics on calls between functions in the application.

The following shows the flat profile pertaining to our sample code :

Saving Results

Saving profiler reports and graphs are recommended before logging out. To save the displayed call graph, use **File -> Screen Dump**.

Xprofiler Menus

MINNESOTA SUPERCOMPUTING INSTITUTE



Flat Profile

<u>File</u>	<u>Code Display</u>		<u>Utility</u>			<u>Help</u>	
%time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name	
34.2	6.07	6.07	101	60.10	60.10	.xsolve [3]	xsol
32.3	11.79	5.72	101	56.63	56.63	.ysolve [4]	ysol
15.8	14.60	2.81	101	27.82	27.82	.yrhs [5]	yrhs
13.6	17.02	2.42	101	23.96	23.96	.xrhs [6]	xrhs
2.0	17.38	0.36	1	360.00	410.00	.setup_matrix [7]	setu
0.7	17.51	0.13				.__mcount [8]	../
0.5	17.59	0.08	90000	0.00	0.00	._exp [10]	../
0.3	17.64	0.05	180000	0.00	0.00	.pot [11]	pot.
0.2	17.67	0.03	1	30.00	110.00	.init_psi [9]	init
0.1	17.68	0.01	4	2.50	2.50	.strspn [16]	../
0.1	17.69	0.01	1	10.00	10.00	.load1 [20]	../
0.1	17.70	0.01	1	10.00	17550.00	.qmfd2 [1]	qmfd
0.1	17.71	0.01				.\$SAVEF24 [21]	nona
0.1	17.72	0.01				._xlfWriteLDChar [22]	ldwr
0.1	17.73	0.01				.qincrement [23]	../
0.0	17.73	0.00	71	0.00	0.00	.xdrmem_getlong [25]	../
0.0	17.73	0.00	68	0.00	0.00	.xdr_long [26]	../
0.0	17.73	0.00	60	0.00	0.00	.__flsbuf [27]	../
0.0	17.73	0.00	60	0.00	0.00	._flsbuf [28]	../
0.0	17.73	0.00	57	0.00	0.00	.leftmost [29]	../

Search Engine: (regular expressions supported)

Xprofiler Menu

MINNESOTA SUPERCOMPUTING INSTITUTE



Source Code for xsolve.f

<u>F</u> ile	<u>U</u> tility		<u>H</u> elp
		no. ticks per line	
line		source code	
20		! solve L*tmp = chi : the forward solve	
21		!-----	
22		do iy=0, nypts-1	
23	1	tmp(0,iy) = chi(0,iy)	
24		do ix=1, nxpts-1	
25	270	tmp(ix,iy) = chi(ix,iy) - subx(ix-1,iy)*tmp(ix-1,iy)	
26		end do	
27		end do	
28			
29			
30		!-----	
31		! solve DLT*phi = tmp : the back solve	
32		!-----	
33		do iy=0, nypts-1	
34	4	psi(nxpts-1,iy) = inv_diagx(nxpts-1,iy)*tmp(nxpts-1,iy)	
35		do ix=nxpts-2, 0, -1	
36	332	psi(ix,iy) = inv_diagx(ix,iy) * tmp(ix,iy)	
37		& - subx(ix,iy) * psi(ix+1,iy)	
38		end do	
39		end do	

Search Engine: (regular expressions supported)

xsolve