

Numascale Best Practice Guide

Atle Vesterkjær
Senior Software Engineer
Numascale
E-mail: av@numascale.com
Website: www.numascale.com

Revision 1.4, May 22, 2015

Revision History

Rev	Date	Author	Changes
0.1	2014-03-03	Atle Vesterkjær	First version
1.0	2014-03-05	Atle Vesterkjær	Added link to wiki. Updated with better fonts.
1.1	2014-03-16	Atle Vesterkjær	Installation hints
1.2	2014-08-28	Atle Vesterkjær	Added information about how to run hybrid applications (MPI+OpenMP)
1.3	2014-08-28	Atle Vesterkjær	Added information about how to run linear algebra code using thread safe scalapack
1.4	2015-05-22	Atle Vesterkjær	Fixed the MPI and Hybrid rankfiles to fit the interpretation of socket in newer kernels, tested on 3.19.5-numascale45+

1	INTRODUCTION	4
1.1	ABSTRACT.....	4
1.2	ABBREVIATIONS.....	4
1.3	BACKGROUND	4
2	MULTITHREADED APPLICATION THAT CAN BENEFIT FROM A LARGER NUMBER OF CORES THEN A SINGLE SERVER CAN PROVIDE	6
2.1	TUNING YOUR APPLICATION FOR A NUMASCALE SYSTEM	7
2.1.1	OpenMP	7
2.1.2	MPI	9
2.1.3	Hybrid (MPI+OpenMP)	11
2.1.4	Pthreads, or other multithreaded applications	12
2.2	RUNNING LINEAR ALGEBRA LIBRARIES	15
3	MEMORY LAYOUT OF APPLICATIONS IN A NUMASCALE SHARED MEMORY SYSTEM 16	
3.1	ALL THREADS ARE AVAILABLE, E.G. THROUGH HTOP	18
3.2	KEEP THREAD LOCAL STORAGE (TLS) LOCAL	22
3.1	USE A NUMA-AWARE MEMORY ALLOCATOR	22
3.1.1	The HOARD memory allocator	22
3.1.1	The NCALLOC memory allocator	22
3.2	ARRANGE YOUR DATA IN A NUMA-AWARE	23
4	COMPILERS	24
5	INSTALLING NUMASCALE BTL	25
6	IPMI	26
6.1	POWER CONTROL	26
6.2	CONSOLE ACCESS	26
7	INSTALLING APPLICATIONS	27
7.1	GROMACS.....	27

1 INTRODUCTION

1.1 Abstract

This is a summary of best practices for achieving optimal results using Numascale Shared Memory Systems with emphasis on running and optimizing applications.

Keywords are:

- OpenMP, affinity, different compilers
- MPI, linpack, NPB, rankfiles
- Vmstat, htop, top, numastat, “*cat/proc/cpuinfo*” etc.
- Numa-aware optimizations of code

1.2 Abbreviations

Throughout this document, we use the following terms:

Numascale AS	The name of the company that develops and owns NumaChip and NumaConnect.
NumaServer	A server equipped with a NumaConnect card.
NumaMaster	A NumaServer with an instance of the operating system installed.

1.3 Background

This document is an addendum to documents at http://www.numascale.com/numa_support.html that cover all aspects of setup, cabling and installation of a Numascale Shared Memory system:

- Numascale hardware installation guides
 - <https://resources.numascale.com/Numascale-Hardware-Installation-IBM-System-x3755-M3.pdf>
 - <https://resources.numascale.com/Numascale-Hardware-Installation-H8QGL-iF+.pdf>
- NumaManager Quick Start Guide, <https://resources.numascale.com/nma-quickstart.pdf>
 - a one page Quick Start Guide for deployment of NumaConnect
- NumaConnect Deployment with NumaManager, <https://resources.numascale.com/nc-deployment-nma.pdf>
 - a complete guide to get a NumaConnect system up and running. Using the NumaManager in this process is strongly recommended and the NumaManager is included in the NumaConnect delivery
- NumaConnect Deployment, <https://resources.numascale.com/nc-deployment.pdf>
 - a complete guide to get a NumaConnect system up and running when no NumaManager is available. It is recommended to use the NumaManager in this process

Any **multithreaded** application can run on Numascale Shared Memory System. The basic criteria's for being interested in a Numascale Shared Memory System are typically:

- You have a multithreaded application that can benefit from a larger number of cores than a single server can provide.
- You have an application that requires more memory than a single server can provide
- You have an application that requires both more cores and more memory than a single server can provide

2 MULTITHREADED APPLICATION THAT CAN BENEFIT FROM A LARGER NUMBER OF CORES THEN A SINGLE SERVER CAN PROVIDE

If you have a multithreaded application that can benefit from a larger number of cores than a single server can provide it is a good candidate for a Numascale Shared Memory System. Compared to other High-End ccNuma solutions and software only ccNuma solutions cores comes at a very low price in a NumaConnect Shared Memory System.

If your application scales well when you use more of the cores in your server, then there is a good chance that it will succeed running on a Numascale Shared Memory System with a larger number of cores than you have in your single server.

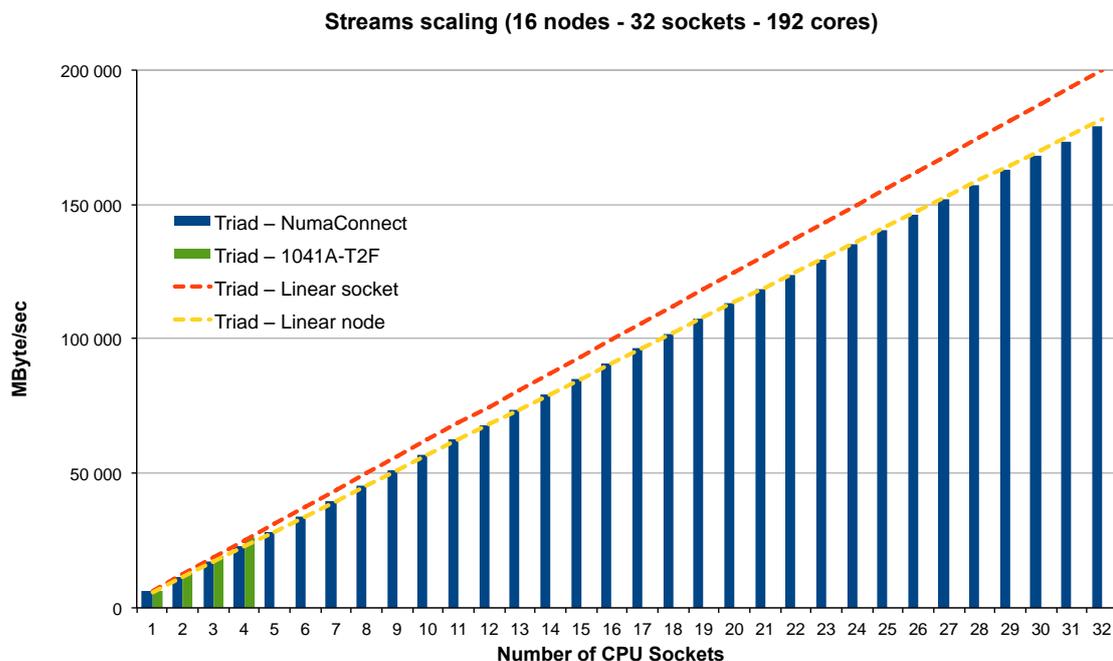


Figure 1: On a Numascale Shared Memory System the Streams benchmark can run in parallel on a larger number of cores than you have in your single server

The Stream benchmark is used as an example of such an application. Looking at the green bars in Figure 1 you see that the speedup increases when using more sockets inside the one server. This makes the application a good candidate for ccNuma systems.

In chapter 3 we will look into how a multithreaded application accesses memory in a ccNuma system. This is very relevant to how well it will scale in a large shared memory system when all CPUs and memory no longer are on the same physical server.

2.1 Tuning your application for a Numascale system

2.1.1 OpenMP

An OpenMP application has very nice runtime characteristics. It is the de-facto state of the art way of doing multithreaded programs. OpenMP pragmas allows you to parallelize code. OpenMP has a very nice way of parallelizing code as it gives the compiler a hint on where to place the parallel threads. Use environment variables in order to control the runtime of applications using OpenMP.

- GNU: OMP_NUM_THREADS=64 GOMP_CPU_AFFINITY="0-255:4" bin/ep.C.x
- GNU: OMP_NUM_THREADS=128 GOMP_CPU_AFFINITY="0-127" bin/ep.C.x
- GNU: OMP_NUM_THREADS=128 GOMP_CPU_AFFINITY="0-255:2" bin/ep.C.x
- PGI: OMP_THREAD_LIMIT=128 OMP_PROCS_BIND=true OMP_NUM_THREADS=128 MP_BIND=yes MP_BLIST=\$(seq -s, 0 2 255) bin_pgftran/ep.C.x

The AMD Opteron™ 6300 series processor is organized as a Multi-Chip Module (two CPU dies in the package), interconnected using a Hyper Transport link. Each die has its own memory controller interface to allow external connection of up to two memory channels per die. This memory interface is shared among all the CPU core pairs on each die. Each core pair appears to the operating system as two completely independent CPU cores. Thus, to the OS, the device shown in Figure 2 appears as 16 CPUs.

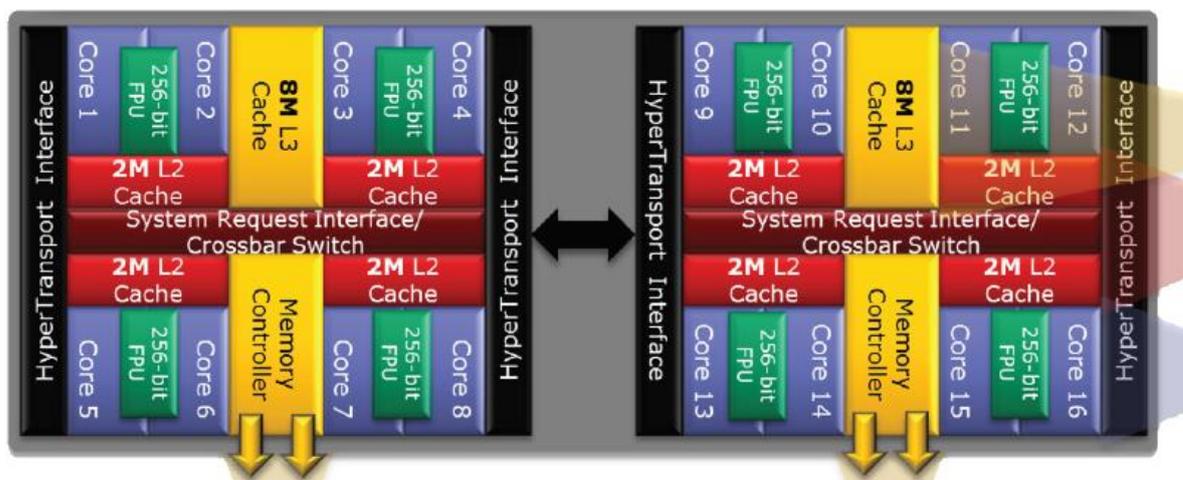


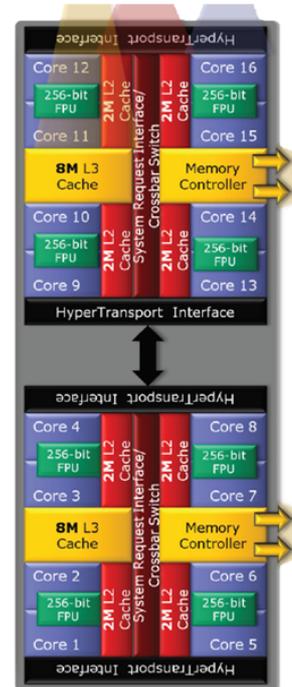
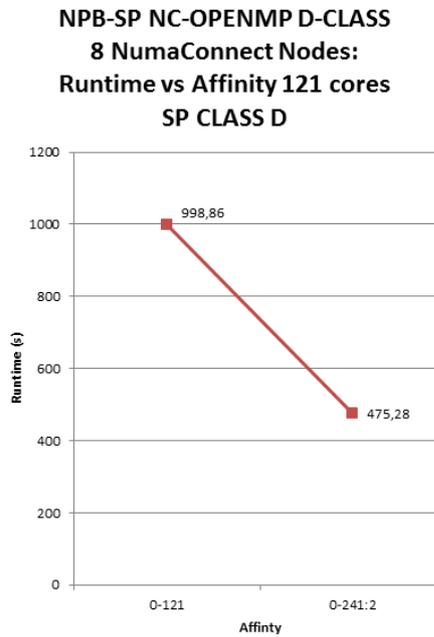
Figure 2: The AMD Opteron™ 6300 series processor is organized as a Multi-Chip Module

In Numascale Shared Memory System with AMD 6380 CPUs the most efficient way to run an FPU intensive OpenMP application with 128 threads in a 256 cores system is:

```
[atle@numademo NPB3.3-OMP]$ OMP_NUM_THREADS=128 GOMP_CPU_AFFINITY="0-255:2" bin/sp.D.x
```

NPB SP D-class runtime

- ▶ The SP D-class using 121 cores has a **speedup of 2.1** when running on 8 NumaConnect Nodes, compared to 4 NumaConnect Nodes
- ▶ The number of cores pr. FPU is one when running on 8 NumaConnect Nodes and two when running on 4 NumaConnect Nodes.
- ▶ The picture on the right shows that the AMD Opteron™ 6300 series processor is organized as a Multi-Chip Module (two CPU dies in the package). Each die has 8 cores, but 4 FPUs



Copyright 2014 All rights reserved.

1

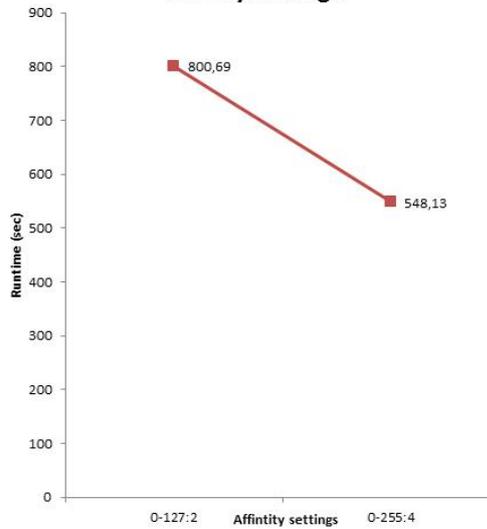
Figure 3: Speedup of Numascale OpenMP application when using more FPUs

If an OpenMP application uses 64 threads, the most efficient way to distribute the threads in the system is to get lower load on the memory interface and L3 cache shared among all the CPU core pairs on each die is using every fourth core in your system:

```
[atle@numademo NPB3.3-OMP]$ OMP_NUM_THREADS=64 GOMP_CPU_AFFINITY="0-255:4" bin/sp.D.x
```

- ▶ The SP D-class using 64 cores has a **speedup of 1,46** when running on 8 NumaConnect Nodes, compared to 4 NumaConnect Nodes
- ▶ **In this test the number of FPUs is constant**
- ▶ The scaling is good when using more nodes
- ▶ The AMD Opteron™ 6300 series memory controller interface allow external connection of up to two memory channels per die. This memory interface is shared among all the CPU core pairs on each die.

NPB-SP NC-OPENMP D-CLASS
Runtime (sec) is shorter when using 8 NumaConnect Nodes and best affinity settings



Copyright 2014 All rights reserved.

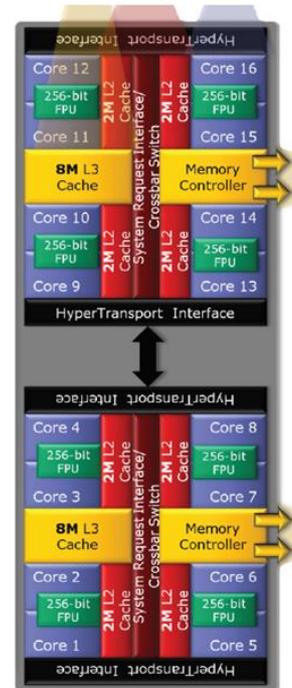


Figure 4: Speedup of Numascale OpenMP application when balancing the load to get a FAT link to memory and L3 cache

2.1.2 MPI

Numascale has developed a byte transfer layer (BTL) for OpenMPI that fully utilizes the NumaConnect. The Numascale OpenMPI implementation is very efficient compared to a standard shared memory BTL. Standard tests show a speedup of 4.5 – 19.9 on typical systems.

MPI_Barrier on 32 Nodes

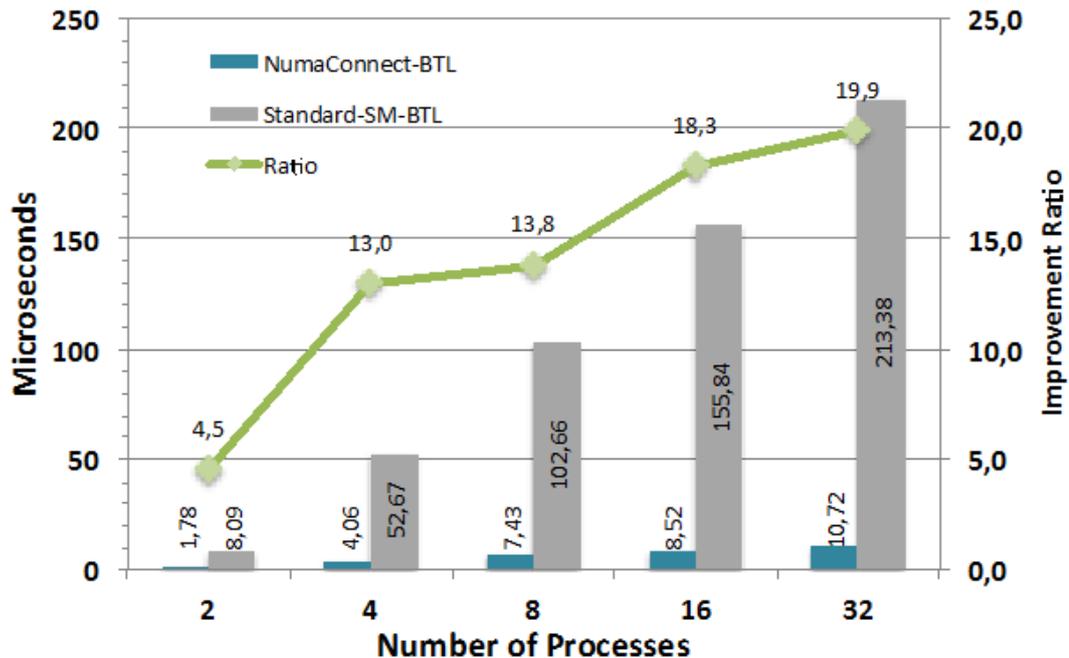


Figure 5: Numascale NumaConnect-BTL boosts your MPI performance

In order to use the NumaConnect-BTL make sure that it is included in the path:

```
[atle@numademo NPB3.3-MPI]$ export LD_LIBRARY_PATH=/opt/openmpi-1.7.2-
numaconnect/lib:$LD_LIBRARY_PATH
[atle@numademo NPB3.3-MPI]$ export PATH=/opt/openmpi-1.7.2-
numaconnect/bin:$PATH
[atle@numademo NPB3.3-MPI]$ which mpiexec
/opt/openmpi-1.7.2-numaconnect/bin/mpiexec
[atle@numademo NPB3.3-MPI]$
```

There are different ways of optimizing an MPI application. In order to avoid using slow disk IO we use tmpfs (dev/shm) as location for temporary OpenMPI files set by the flag `OMPI_PREFIX_ENV`.

Use the `--bind-to-core` option for jobs running on all the cores in the system:

```
[atle@numademo ~]$ OMPI_PREFIX_ENV=/dev/shm mpiexec -n 256 --bind-to-core
-mca btl self,nc bin/lu.D.256
```

Use rankfiles when not running on all cores in the system.

An MPI application that utilizes less than all cores in a Numascale Shared Memory is optimized using rank files. In a 256 core system using AMD 6380 CPUs you can instruct your application to use one FPU per core by selecting every second core in a rankfile:

```
[atle@numademo NPB3.3-MPI]$ cat rank128_lu
rank 0=numademo slot=0:0
rank 1=numademo slot=0:2
```

```
rank 2=numademo slot=0:4
rank 3=numademo slot=0:6
... until
rank 120=numademo slot=15:0
rank 121=numademo slot=15:2
rank 122=numademo slot=15:4
rank 123=numademo slot=15:6
rank 124=numademo slot=15:8
rank 125=numademo slot=15:10
rank 126=numademo slot=15:12
rank 127=numademo slot=15:14
atle@numademo NPB3.3-MPI]$
```

By using every fourth core in the system, it is possible to distribute the processes even more utilize all memory controllers and L3 caches in the system:

```
[atle@numademo NPB3.3-MPI]$ cat rank64
rank 0=numademo slot=0:0
rank 1=numademo slot=0:4
rank 2=numademo slot=0:8
rank 3=numademo slot=0:12
...etc until
rank 60=numademo slot=15:0
rank 61=numademo slot=15:4
rank 62=numademo slot=15:8
rank 63=numademo slot=15:12
```

Start the application with these options set:

```
[atle@numademo NPB3.3-MPI]$ OMPI_PREFIX_ENV=/dev/shm mpiexec
-n 64 -rf rank64 -mca btl self,nc bin/lu.D.64
```

2.1.3 Hybrid (MPI+OpenMP)

Applications suites that comes with a hybrid version are able to run “out of the box” on a Numascale Shared Memory System.

Here is one example using the NAS Parallel Benchmarks, NPB-MZ-MPI applications suite that comes with a hybrid version. In this example each of the 16 MPI processes spawns two (2) threads using OpenMP, utilizing 32 cores.

```
[atle@numademo NPB3.3-MZ-MPI]$
export LD_PRELOAD=/opt/HOARD/libhoard.so
[atle@numademo NPB3.3-MZ-MPI]$
export OMP_NUM_THREADS=2;/opt/openmpi-1.6.0-numaconnect/bin/mpirun -n
16 --mca btl nc,self -rf rank16_colon2 bin_gnu/lu-mz.C.16
```

Here is the hybrid rank file used for the test:

```
[atle@numademo NPB3.3-MZ-MPI]$ cat rank16_2
```

```
rank 0=localhost slot=0:0,0:2
rank 1=localhost slot=0:4,0:6
rank 2=localhost slot=0:8,0:10
rank 3=localhost slot=0:12,0:14
...
rank 15=localhost slot=3:8,3:10
```

If you want to run a test on 16 MPI processes each spawning 8 OpenMP threads, in total 128 cores you can use this rank-file:

```
[atle@numademo NPB3.3-MZ-MPI]$ cat rank16_8
```

```
rank 0=h8qgl slot=0:0,0:2,0:4,0:6,0:8,0:10,0:12,0:14
rank 1=h8qgl slot=1:0,1:2,1:4,1:6,1:8,1:10,1:12,1:14
rank 2=h8qgl slot=2:0,2:2,2:4,2:6,2:8,2:10,2:12,2:14
rank 3=h8qgl slot=3:0,3:2,3:4,3:6,3:8,3:10,3:12,3:14
rank 4=h8qgl slot=4:0,4:2,4:4,4:6,4:8,4:10,4:12,4:14
rank 5=h8qgl slot=5:0,5:2,5:4,5:6,5:8,5:10,5:12,5:14
rank 6=h8qgl slot=6:0,6:2,6:4,6:6,6:8,6:10,6:12,6:14
rank 7=h8qgl slot=7:0,7:2,7:4,7:6,7:8,7:10,7:12,7:14
rank 8=h8qgl slot=8:0,8:2,8:4,8:6,8:8,8:10,8:12,8:14
rank 9=h8qgl slot=9:0,9:2,9:4,9:6,9:8,9:10,9:12,9:14
rank 10=h8qgl slot=10:0,10:2,10:4,10:6,10:8,10:10,10:12,10:14
rank 11=h8qgl slot=11:0,11:2,11:4,11:6,11:8,11:10,11:12,11:14
rank 12=h8qgl slot=12:0,12:2,12:4,12:6,12:8,12:10,12:12,12:14
rank 13=h8qgl slot=13:0,13:2,13:4,13:6,13:8,13:10,13:12,13:14
rank 14=h8qgl slot=14:0,14:2,14:4,14:6,14:8,14:10,14:12,14:14
rank 15=h8qgl slot=15:0,15:2,15:4,15:6,15:8,15:10,15:12,15:14
and run:
```

```
[atle@numademo NPB3.3-MZ-MPI]$ export
```

```
OMP_NUM_THREADS=8;/opt/openmpi-1.6.0-numaconnect/bin/mpirun -n 16
--mca btl nc,self -rf rank16_8 bin_gnu/lu-mz.C.16
```

2.1.4 Pthreads, or other multithreaded applications

taskset or numactl apply a mask for the creation of threads in a multithreaded program. Applications written with pthreads can use this to choose cores and memory segments for execution. In tests at Numascale, taskset and numactl perform equally well, but other online resources recommend using numactl for guaranteed memory binding.

numascale

```
[root@x3755 02_BWA_MEM]# taskset
taskset (util-linux-ng 2.17.2)
usage: taskset [options] [mask | cpu-list] [pid | cmd [args...]]
set or get the affinity of a process
```

```
p, --pid                operate on existing given pid
c, --cpu-list           display and specify cpus in list format
h, --help              display this help
V, --version           output version information
```

The default behavior is to run a new command:

```
taskset 03 sshd -b 1024
```

You can retrieve the mask of an existing task:

```
taskset -p 700
```

Or set it:

```
taskset -p 03 700
```

List format uses a comma-separated list instead of a mask:

```
taskset -pc 0,3,7-11 700
```

Ranges in list format can take a stride argument:

e.g. 0-31:2 is equivalent to mask 0x55555555

```
[root@x3755 02_BWA_MEM]# numactl
```

```
usage: numactl [--interleave=nodes] [--preferred=node]
```

```
 [--physcpubind=cpus] [--cpunodebind=nodes]
```

```
 [--membind=nodes] [--localalloc] command args ...
```

```
numactl [--show]
```

```
numactl [--hardware]
```

```
numactl [--length length] [--offset offset] [--shmmode shmmode]
```

```
 [--strict]
```

```
 [--shmid id] --shm shmkeyfile | --file tmpfsfile
```

```
 [--huge] [--touch]
```

```
memory policy | --dump | --dump-nodes
```

memory policy is `interleave`, `preferred`, `membind`, `localalloc`

`nodes` is a comma delimited list of node numbers or A-B ranges or all.

`cpus` is a comma delimited list of cpu numbers or A-B ranges or all

all ranges can be inverted with `!`

all numbers and ranges can be made `cpuset-relative` with `+`

the old `cpubind` argument is deprecated.

use `cpunodebind` or `physcpubind` instead

`length` can have `g` (GB), `m` (MB) or `k` (KB) suffixes

```
[root@x3755 02_BWA_MEM]#
```

“numactl `--hardware`” gives a good view of the hardware setup. Here a Numascale Shared Memory System with two NumaServers, 8 sockets and 64 cores is used:

```
[root@x3755 02_BWA_MEM]# numactl--hardware
```

```
available: 8 nodes (0-7)
```

```
node 0 cpus: 0 1 2 3 4 5 6 7
```

```
node 0 size: 65491 MB
```

```

node 0 free: 24474 MB
node 1 cpus: 8 9 10 11 12 13 14 15
node 1 size: 65536 MB
node 1 free: 56116 MB
node 2 cpus: 16 17 18 19 20 21 22 23
node 2 size: 65536 MB
node 2 free: 64338 MB
node 3 cpus: 24 25 26 27 28 29 30 31
node 3 size: 47104 MB
node 3 free: 46263 MB
node 4 cpus: 32 33 34 35 36 37 38 39
node 4 size: 65536 MB
node 4 free: 64358 MB
node 5 cpus: 40 41 42 43 44 45 46 47
node 5 size: 65536 MB
node 5 free: 62861 MB
node 6 cpus: 48 49 50 51 52 53 54 55
node 6 size: 65536 MB
node 6 free: 63288 MB
node 7 cpus: 56 57 58 59 60 61 62 63
node 7 size: 49152 MB
node 7 free: 46946 MB
node distances:
node  0  1  2  3  4  5  6  7
0:  10  16  16  22  100  100  100  100
1:  16  10  16  22  100  100  100  100
2:  16  16  10  16  100  100  100  100
3:  22  22  16  10  100  100  100  100
4:  100  100  100  100  10  16  16  22
5:  100  100  100  100  16  10  16  22
6:  100  100  100  100  16  16  10  16
7:  100  100  100  100  22  22  16  10

```

Here are some advices on how to tune a multithreaded application by selecting a runtime mask:

➤ **Running only on the first socket**

```

taskset -c 0-7 /scratch/demouser04/04_Software/bwa-0.7.4/bwa mem -t
$procs -t ${REF} ${INP} ${INP2}
numactl-cpunodebind 0,1,2,3 --mbind 0,1,2,3
/scratch/demouser04/04_Software/bwa-0.7.4/bwa mem -t $procs -t ${REF}
${INP} ${INP2}

```

➤ **Running on all cores in the first NumaServer**

```

taskset -c 0-31 /scratch/demouser04/04_Software/bwa-0.7.4/bwa mem -t
$procs -M ${REF} ${INP} ${INP2}
numactl-cpunodebind 0,1,2,3,4,5,6,7 --mbind 0,1,2,3,4,5,6,7
/scratch/demouser04/04_Software/bwa-0.7.4/bwa mem -t $procs -t ${REF}
${INP} ${INP2}
numactl-physcpubind
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27

```

```
,28,29,30,31 --membind 0,1,2,3,4,5,6,7  
/scratch/demouser04/04_Software/bwa-0.7.4/bwa mem -t $procs -t ${REF}  
${INP} ${INP2}
```

➤ **Running on all FPU's in the first NumaServer**

```
taskset -c 0-31:2 /scratch/demouser04/04_Software/bwa-0.7.4/bwa mem -t  
$procs -M ${REF} ${INP} ${INP2}  
numactl-physicspubind 0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30 --membind  
0,1,2,3,4,5,6,7 /scratch/demouser04/04_Software/bwa-0.7.4/bwa mem -t  
$procs -t ${REF} ${INP} ${INP2}
```

➤ **Running on all FPU's in the system**

```
taskset -c 0-63:2 /scratch/demouser04/04_Software/bwa-0.7.4/bwa mem -t  
$procs -M ${REF} ${INP} ${INP2}  
numactl-physicspubind 0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30 --membind  
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15  
/scratch/demouser04/04_Software/bwa-0.7.4/bwa mem -t $procs -t ${REF}  
${INP} ${INP2}
```

Monitor the system usage with `vmstat`, `htop`, `top` and `numastat` to see if it behaves as intended with these runtime parameters. Please note that `htop` and `top` requires some system resources and might influence the performance. We recommend restarting the benchmark with the more lightweight `vmstat` for the final benchmark test when the runtime parameters have been qualified.

2.2 Running linear algebra libraries

ScaLAPACK is a scalable subset of the LAPACK (Linear Algebra Package) routines. It is open source and can be found at www.netlib.org. All commonly used linear algebra libraries like IBM ESSL, Intel MKL, AMD ADSL, IMSL, NAG use LAPACK and BLAS. The vendors supply a highly optimized version of the Netlib code. Numerical applications like Matlab or Fluent use these optimized libraries.

ScaLapack is based on MPI communication in Blacs and is not thread-safe.

Our hardware supports solvers running threads distributed. We can provide a thread-safe ScaLapack and OpenMP, shared memory based Blacs library. This way the solvers run in a single process using OpenMP distributed on several machines.

Please check out <https://wiki.numascale.com/tips/building-openmp-based-scalapack> for more information.

3 MEMORY LAYOUT OF APPLICATIONS IN A NUMASCALE SHARED MEMORY SYSTEM

In order to illustrate the basic difference between a Numascale Shared Memory System and a cluster we consider a large job that can be solved in parallel by a large number of resources. In the clustering world, a large data set will be split into smaller fragment that will be distributed to individual nodes that are loosely coupled using the MPI protocol.

Imagine that such a job represents a large image like in Figure 6.

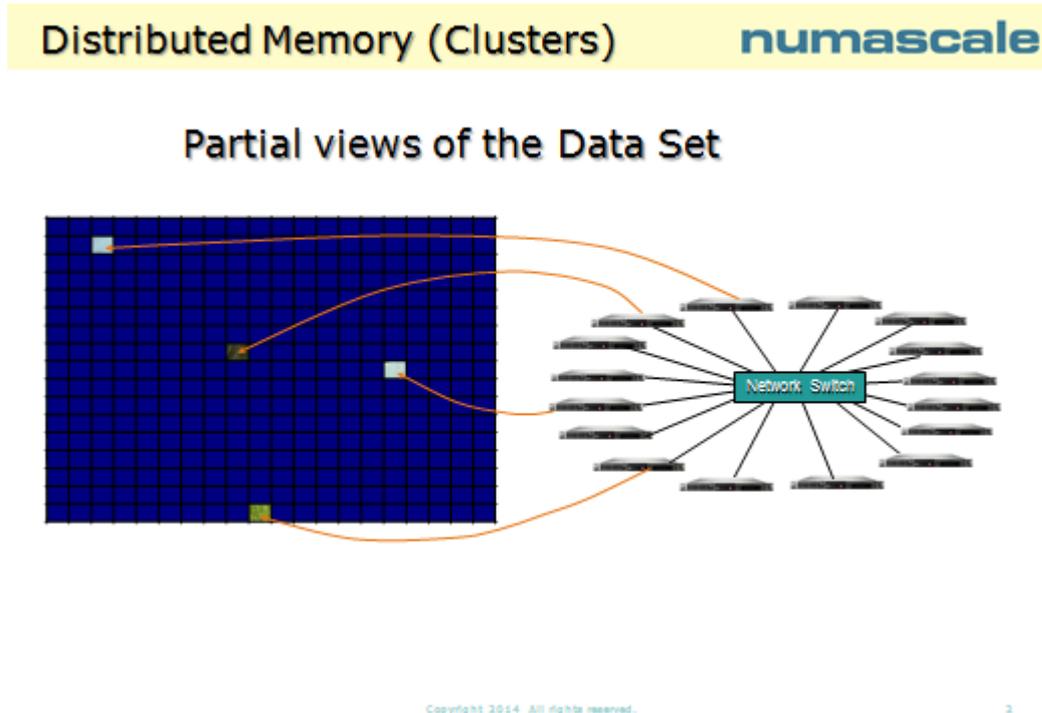
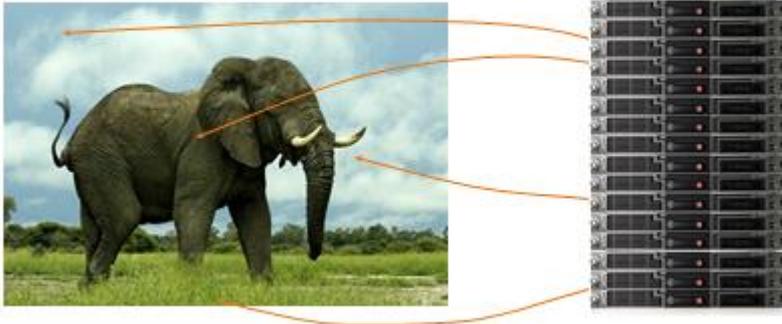


Figure 6: Distributed Memory (Clusters)

Consider the same job in the Numascale context. A shared view of the whole dataset is just as simple as on a standalone server.

Shared view of the Entire Data Set



No Virtualization SW Required

Copyright 2014. All rights reserved.

3

Figure 7: Shared Memory

Program a Numascale Shared Memory System just as a standalone server. The full memory range is available to all applications. You can run "top" on a 1.5 TB system.

```
top - 09:29:57 up 20 days, 20:07, 6 users, load average: 8.32, 8.31, 8.32
Tasks: 3068 total, 3 running, 3065 sleeping, 0 stopped, 0 zombie
Cpu(s): 1.4%us, 0.1%sy, 0.0%ni, 98.5%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 1583148160k total, 1185082348k used, 398065812k free, 4k buffers
Swap: 33046524k total, 0k used, 33046524k free, 4035576k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
202867	root	20	0	1119g	1.1t	524	R	815	74.1	3758:34	x.mod2as
197024	root	20	0	19168	8560	1204	R	37	0.0	403:40.55	htop
206397	root	20	0	13512	3704	924	R	14	0.0	0:04.48	top
197023	root	20	0	13512	3700	924	S	13	0.0	177:20.24	top
1399	root	20	0	0	0	0	S	3	0.0	1:03.79	ksoftirqd/348
2187	root	20	0	0	0	0	S	3	0.0	0:50.98	ksoftirqd/545
10	root	20	0	0	0	0	S	2	0.0	290:26.05	rcu_sched
11190	root	20	0	0	0	0	S	1	0.0	68:02.65	kworker/40:1
11177	root	20	0	0	0	0	S	1	0.0	173:47.85	kworker/24:1
11241	root	20	0	0	0	0	S	1	0.0	86:23.99	kworker/44:1
11668	root	20	0	0	0	0	S	1	0.0	53:37.71	kworker/348:1
11688	root	20	0	0	0	0	S	1	0.0	59:31.14	kworker/336:1
12035	root	20	0	0	0	0	S	1	0.0	20:37.15	kworker/545:1
11197	root	20	0	0	0	0	S	0	0.0	96:09.80	kworker/36:1
11203	root	20	0	0	0	0	S	0	0.0	148:15.71	kworker/32:1
11209	root	20	0	0	0	0	S	0	0.0	166:31.05	kworker/28:1
11233	root	20	0	0	0	0	S	0	0.0	82:54.52	kworker/48:1
11626	root	20	0	0	0	0	S	0	0.0	32:13.59	kworker/308:1
11710	root	20	0	0	0	0	S	0	0.0	26:04.65	kworker/357:1
11714	root	20	0	0	0	0	S	0	0.0	31:09.39	kworker/354:1
47717	root	20	0	7764	576	484	S	0	0.0	22:06.24	tail
47736	root	20	0	7764	576	484	S	0	0.0	22:13.57	tail

Figure 8: top on 1.5 TB system

3.1 All threads are available, e.g. through htop

```

1  [|||||] 45.7% 37 [|||||] 28.1% 73 [|||||] 28.1% 109 [|||||] 28.6%
2  [|||||] 100.0% 38 [|||||] 100.0% 74 [|||||] 100.0% 110 [|||||] 100.0%
3  [|||||] 28.5% 39 [|||||] 28.1% 75 [|||||] 28.1% 111 [|||||] 28.5%
4  [|||||] 28.5% 40 [|||||] 28.0% 76 [|||||] 28.1% 112 [|||||] 28.6%
5  [|||||] 28.5% 41 [|||||] 28.0% 77 [|||||] 28.1% 113 [|||||] 28.6%
6  [|||||] 28.6% 42 [|||||] 28.1% 78 [|||||] 28.0% 114 [|||||] 28.6%
7  [|||||] 27.0% 43 [|||||] 27.5% 79 [|||||] 27.0% 115 [|||||] 27.5%
8  [|||||] 100.0% 44 [|||||] 100.0% 80 [|||||] 100.0% 116 [|||||] 53.3%
9  [|||||] 27.1% 45 [|||||] 27.5% 81 [|||||] 27.1% 117 [|||||] 27.5%
10 [|||||] 27.1% 46 [|||||] 27.5% 82 [|||||] 27.1% 118 [|||||] 100.0%
11 [|||||] 27.0% 47 [|||||] 27.5% 83 [|||||] 27.1% 119 [|||||] 27.6%
12 [|||||] 27.0% 48 [|||||] 27.5% 84 [|||||] 27.1% 120 [|||||] 27.5%
13 [|||||] 79.0% 49 [|||||] 77.5% 85 [|||||] 77.9% 121 [|||||] 79.0%
14 [|||||] 79.0% 50 [|||||] 100.0% 86 [|||||] 100.0% 122 [|||||] 100.0%
15 [|||||] 79.0% 51 [|||||] 77.9% 87 [|||||] 77.9% 123 [|||||] 79.0%
16 [|||||] 79.0% 52 [|||||] 77.5% 88 [|||||] 77.5% 124 [|||||] 79.1%
17 [|||||] 79.0% 53 [|||||] 77.5% 89 [|||||] 77.5% 125 [|||||] 79.1%
18 [|||||] 100.0% 54 [|||||] 77.5% 90 [|||||] 77.5% 126 [|||||] 79.1%
19 [|||||] 61.0% 55 [|||||] 62.0% 91 [|||||] 62.3% 127 [|||||] 61.2%
20 [|||||] 100.0% 56 [|||||] 62.0% 92 [|||||] 100.0% 128 [|||||] 61.2%
21 [|||||] 61.3% 57 [|||||] 100.0% 93 [|||||] 62.0% 129 [|||||] 61.2%
22 [|||||] 61.0% 58 [|||||] 62.0% 94 [|||||] 62.3% 130 [|||||] 100.0%
23 [|||||] 61.3% 59 [|||||] 62.0% 95 [|||||] 62.3% 131 [|||||] 61.2%
24 [|||||] 61.0% 60 [|||||] 62.0% 96 [|||||] 62.0% 132 [|||||] 61.2%
25 [|||||] 57.5% 61 [|||||] 60.0% 97 [|||||] 60.5% 133 [|||||] 57.2%
26 [|||||] 100.0% 62 [|||||] 100.0% 98 [|||||] 60.5% 134 [|||||] 57.2%
27 [|||||] 57.5% 63 [|||||] 60.0% 99 [|||||] 60.5% 135 [|||||] 100.0%
28 [|||||] 57.5% 64 [|||||] 60.0% 100 [|||||] 60.5% 136 [|||||] 57.2%
29 [|||||] 57.5% 65 [|||||] 60.0% 101 [|||||] 60.5% 137 [|||||] 57.5%
30 [|||||] 57.5% 66 [|||||] 60.3% 102 [|||||] 100.0% 138 [|||||] 57.2%
31 [|||||] 37.0% 67 [|||||] 37.2% 103 [|||||] 38.0% 139 [|||||] 36.8%
32 [|||||] 100.0% 68 [|||||] 100.0% 104 [|||||] 100.0% 140 [|||||] 100.0%
33 [|||||] 37.0% 69 [|||||] 37.2% 105 [|||||] 38.0% 141 [|||||] 36.8%
34 [|||||] 37.0% 70 [|||||] 37.2% 106 [|||||] 38.0% 142 [|||||] 36.8%
35 [|||||] 37.0% 71 [|||||] 37.2% 107 [|||||] 38.0% 143 [|||||] 36.6%
36 [|||||] 37.2% 72 [|||||] 37.2% 108 [|||||] 38.0% 144 [|||||] 36.8%
Mem[|||||] 52118/384880MB Tasks: 54, 130 thr; 144 running
Swp[|||||] 0/0MB Load average: 49.64 13.40 4.74
Uptime: 00:05:35

```

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
10635	root	20	0	2600M	2117M	4128	R	636.0	0.6	2:55.09	Linux_AMD64_ACML440_MP/xhpl /tmp/HPL.dat.zk0RhF
10623	root	20	0	2613M	2102M	4156	R	636.0	0.5	2:50.99	Linux_AMD64_ACML440_MP/xhpl /tmp/HPL.dat.zk0RhF
10629	root	20	0	2613M	2115M	4124	R	634.0	0.5	2:52.77	Linux_AMD64_ACML440_MP/xhpl /tmp/HPL.dat.zk0RhF
10641	root	20	0	2588M	2094M	4124	R	633.0	0.5	2:57.50	Linux_AMD64_ACML440_MP/xhpl /tmp/HPL.dat.zk0RhF
10636	root	20	0	2627M	2129M	4152	R	561.0	0.6	2:51.58	Linux_AMD64_ACML440_MP/xhpl /tmp/HPL.dat.zk0RhF
10642	root	20	0	2602M	2112M	4168	R	557.0	0.5	2:56.39	Linux_AMD64_ACML440_MP/xhpl /tmp/HPL.dat.zk0RhF
10630	root	20	0	2627M	2130M	4180	R	556.0	0.6	2:51.86	Linux_AMD64_ACML440_MP/xhpl /tmp/HPL.dat.zk0RhF

Figure 9: htop on 144 cores

The major difference from a single CPU server and a Numascale Shared Memory System is that there are differences in the memory latency. In a ccNuma system, the memory latency differs depending on where the actual memory is located.

Thread latency (remote non-polluting write latency) for an arbitrary numanode (socket no 2) when accessing all the other numanodes (sockets 1-32) using 8 NumaServers each equipped with...

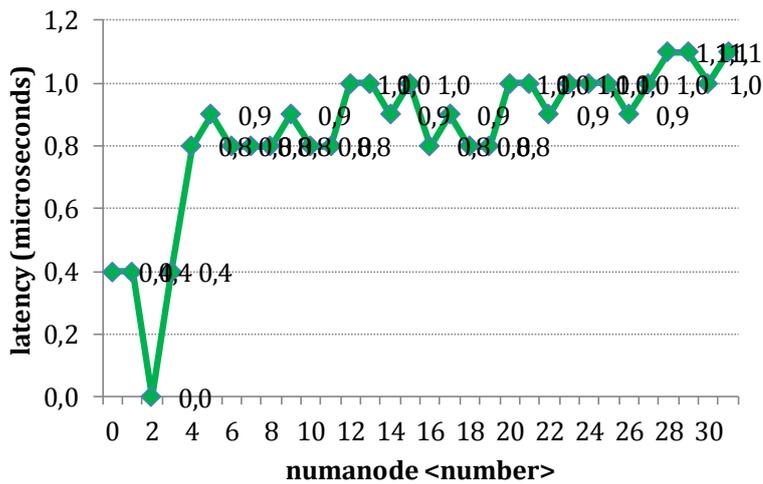


Figure 10: Memory latencies in a Numascale system

Although a ccNuma system is logically similar to a traditional SMP, it is physically different. The runtime of an application is very dependent on where the data is located relative to the running thread.

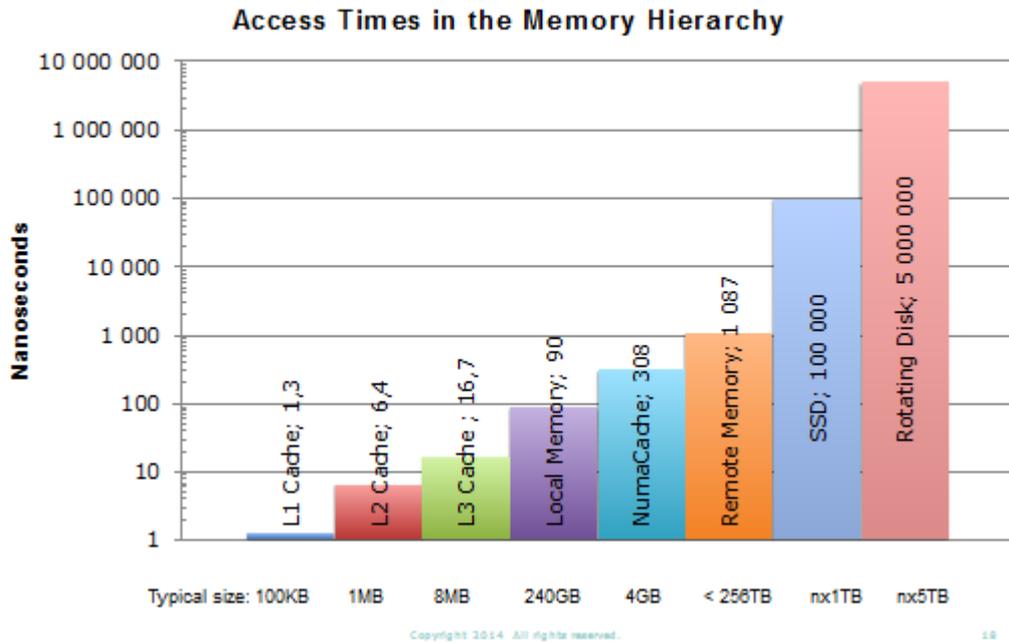


Figure 11: Memory hierarchy in a Numascale system

“numastat” monitor the memory access pattern in a system and can be used to verify the runtime the runtime parameters.

```
[atle@x3755 ~]$ numastat
```

	node0	node1	node2	node3
numa_hit	116198855	22674225	1601598	326121
numa_miss	0	390637	0	0
numa_foreign	390637	0	0	0
interleave_hit	26017	26221	26298	26107
local_node	116197682	22647868	1576067	299901
other_node	1173	416994	25531	26220

	node4	node5	node6	node7
numa_hit	2484895	4957523	4425252	25191604
numa_miss	0	0	0	0
numa_foreign	0	0	0	0
interleave_hit	26026	26209	26289	26088
local_node	2458717	4931194	4398832	25165424
other_node	26178	26329	26420	26180

```
[atle@x3755 ~]$
```

“numastat” can also be used to monitor the memory access pattern for a running process.

```
[atle@x3755 ~]$ ps -elf | grep bwa
4 S root      25459 25458 99  80   0 - 1274150 futex_ 10:29 pts/3  00:36:30
/home/demouser04/04_Software/bwa-0.7.4/bwa mem -t 128 -M
../../../../00_RAW_DATA/EC_K12
../../../../00_RAW_DATA/QBICECKH100001W4_GCCAAT_L001_R2_001.fastq
[root@x3755 atle]# numastat -p 25459
Per-node process memory usage (in MBs) for PID 25459 (bwa)
Node 0          Node 1          Node 2
```

numascale

Huge	0.00	0.00	0.00
Heap	0.00	0.00	0.00
Stack	0.18	0.09	0.19
Private	63.66	38.99	43.42

Total	63.85	39.07	43.61

	Node 3	Node 4	Node 5

Huge	0.00	0.00	0.00
Heap	0.00	0.00	0.00
Stack	0.18	2.52	0.38
Private	31.44	82.00	66.58

Total	31.62	84.51	66.96

	Node 6	Node 7	Total

Huge	0.00	0.00	0.00
Heap	0.00	0.00	0.00
Stack	11.33	0.46	15.32
Private	3434.12	64.36	3824.57

Total	3445.45	64.82	3839.89

[root@x3755 atle]#

Applications that take the different memory latencies in a ccNUMA system into account scales well on a Numascale Shared Memory System. The RWTH TrajSearch code has the **most speedup on NumaConnect**, even if it was originally adapted to Scale MP.

JARA|HPC

AN INITIATIVE OF



Scaling OpenMP programs to thousand cores on the Numascale architecture

Numascale

Numascale is a European SME specializing in interconnect for high performance and enterprise computing. The differentiator for Numascale's interconnect is the shared memory and cache coherency mechanisms. These features allow programs to access any memory location and any memory mapped I/O device in a multiprocessor system with high degree of efficiency. It provides scalable systems with a unified programming model that stays the same from the small multi-core machines used in laptops and desktops to the largest imaginable single system image machines that may contain thousands of processors. The architecture is commonly classified as ccNUMA or Numa but the interconnect system can alternatively be used as low latency clustering interconnect. The technology comes as an add on card to standard servers providing large shared memory at cluster price.

The Oslo System



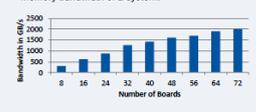
The University of Oslo's Center for Information Technology, USIT feature this Numascale installation. The systems is a PRACE prototype and has been used for some of the work between Numascale and JARA/HPC.

- 72 IBM x3755 M3 nodes
- 144 AMD 6174 CPUs
- 1728 Cores
- 4,6TB Memory

Memory Bandwidth

Memory bandwidth is essential for many scientific applications.

- Every processor has its own memory and adds to the total memory bandwidth.
- Stream [1] is a standard benchmark to evaluate the memory bandwidth of a system.

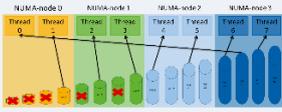


Optimizations for large NUMA systems

- Reduced Synchronization Overhead:
 - local data copies where possible
 - local buffers for synchronize writes
 - multithreaded memory allocator, `kmp_malloc`
 - thread local random number generator
- NUMA aware memory placement
- usage of the Adaptive NUMA Scheduler to handle load imbalance and maintain most possible data locality

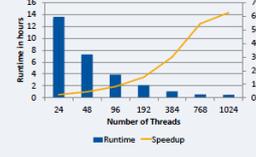


The adaptive NUMA Scheduler



- Combines principle of static and dynamic scheduling.
- Static layout defines how work should be distributed.
- Idle threads take work from the threads with least progress.
- To reduce interference work of foreign threads will get iterations from the highest index backwards.

Performance Results



TrajSearch

TrajSearch is a code to investigate turbulences which occur during combustion. It is a post-processing code for dissipation element analysis developed by Peters and Wang [2]. It decomposes a highly resolved three dimensional turbulent flow field obtained by Direct Numerical Simulation (DNS) into non-arbitrary, space-filling and non-overlapping geometrical elements called 'dissipation elements'. Starting from every grid point in the direction of ascending and descending gradient of an underlying diffusion controlled scalar field, a local maximum, respectively minimum point is found. A dissipation element is defined as a volume from which all trajectories reach the same minimum and maximum point. The dissipation element analysis provides a deeper understanding of turbulence and can be employed to reconstruct important statistical properties as has been shown by Gampert and Göbbert [3].



Summary

- The investigated Numascale system combines 1728 cores and 4.5 TB of main memory in a shared memory machine.
- The accumulated memory bandwidth rises with the number of nodes to over 2 TB/s.
- The application TrajSearch was optimized to run on large NUMA systems by reducing synchronization, optimizing the memory layout and using the Adaptive NUMA Scheduler for work scheduling.
- Overall the code delivers a speedup of 625 running with 1024 threads.

References

[1] John D. McCalpin, STREAM: Sustainable memory bandwidth in high performance computers, <http://www.cs.virginia.edu/stream/>, 1995.

[2] H. Peters and L. Wang, Dissipation element analysis of scalar fields in turbulence, C. R. Mechanique, 334:463-500, 2006.

[3] Markus Gampert, Jens Henrik Göbbert, Michael Gauding, Philip Schäfer, and Norbert Peters, Extensive strain along gradient trajectories in the turbulent kinetic energy field, New Journal of Physics, 2011.

Contact: Dirk Schmidt, schmidt@ic.rwth-aachen.de, Jara, RWTH Aachen University, Germany, Able Vesterkjær, av@numascale.com, Numascale AS

Figure 12: RWTH TrajSearch code

3.2 Keep Thread Local Storage (TLS) local

Numascale has patched the gcc and open64 OpenMP implementation to make sure that the stack and Thread Local Storage (TLS) is local to the running threads. This favors these compilers for OpenMP code. For more information about this feature, see <https://wiki.numascale.com/tips/placing-stack-and-tls-memory> .

Numascale Wiki

Home **Tips** Recommended configurations Recipes FAQ

10 ▾

Title	Author	Hits
Building thread-safe ScaLAPACK	Written by Stefan Olbrich	Hits: 34
Placing stacks and TLS memory	Written by Stefan Olbrich	Hits: 100
The NCALLOC memory allocator	Written by Stefan Olbrich	Hits: 59
Kernel patches	Written by Daniel J Blueman	Hits: 61
OpenMPI using NumaConnect BTL	Written by Stefan Olbrich	Hits: 82
OS tips	Written by Daniel J Blueman	Hits: 109
Run-time tips	Written by Daniel J Blueman	Hits: 136
Compiler tips	Written by Daniel J Blueman	Hits: 157

Figure 13: <https://wiki.numascale.com/tips> show best practice hints

3.1 Use a Numa-aware Memory allocator

3.1.1 The HOARD memory allocator

Numascale is experimenting with other memory allocators than libc in order to speed up the allocation process by not shifting around blocks between threads.

<http://www.hoard.org/> reduces lock contention by using multiple heaps and avoids false sharing in its allocations.

```
export LD_PRELOAD=/opt/HOARD/libhoard.so
```

3.1.1 The NCALLOC memory allocator

Numascale has written its own memory allocator to ensure that the HEAP of a running thread is local to the core it is running on. You can experiment with both the NCALLOC and HOARD memory allocators. You can find more information about the NCALLOC memory allocator in <https://wiki.numascale.com/tips/the-ncalloc-memory-allocator>.

```
export LD_PRELOAD=/opt/HOARD/libncalloc.so
```

3.2 Arrange your data in a Numa-aware

The least efficient memory model for an application that you would like to scale well on ccNuma system is to have one big array of data on one node that all threads has to access. Try to distribute the data as much as possible in order to be able for threads to use memory local to the core they are running on.

4 COMPILERS

Check out <https://wiki.numascale.com/tips> for public tips from Numascale.

AMD has a good guide at <http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/CompilerOptQuickRef-63004300.pdf> for general compiler optimization flags for the AMD Opteron 6300 series.

AMD has done some work in optimizing compiler flags well-known applications:

- <http://www.amd.com/us/products/server/benchmarks/Pages/gromacs-cluster-perf-d-dppc-two-socket-servers.aspx>
- <http://www.amd.com/us/products/server/benchmarks/Pages/high-performance-linpack-cluster-perf.aspx>
- <http://www.amd.com/us/products/server/benchmarks/Pages/molecular-dynamics-namd-cluster-performance-two-socket-servers.aspx>

More can be found at <http://www.amd.com/us/products/server/benchmarks/>

The most common compilers used with the Numascale architecture are:

- GCC 4.8.1
- PGI compiler 13.3
- Open64 compiler (x86_open64-4.5.2.1)
- Intel compiler (yes this works for AMD systems as well)

Numascale has patched the gcc and open64 OpenMP implementation to make sure that the stack and Thread Local Storage (TLS) is local to the running threads. This favors these compilers for OpenMP code.

Please check out:

http://developer.amd.com/wordpress/media/2012/10/51803A_OpteronLinuxTuningGuide_SCREEN.pdf

5 INSTALLING NUMASCALE BTL

Download the source at:

<https://resources.numascale.com/software/openmpi-1.7.2.tar.gz>

It might be helpful to download:

<https://resources.numascale.com/software/openmpi-tools.tar.gz>

Remember to install numactl and numactl-devel using yum first.
Here are the installation steps on my system:

```
tar -zxvf ../openmpi-1.7.2.tar.gz
cd openmpi-tools/
cd automake-1.12.2/
./configure ; make; sudo make install
cd ..
cd autoconf-2.69/
./configure ; make; sudo make install
cd ../
cd flex-2.5.35/
sudo yum install help2ma
./configure ; make; sudo make install
cd ..
cd libtool-2.4.2/
./configure ; make; sudo make install
cd ../m4-1.4.16/
./configure ; make; sudo make install
cd ../../openmpi-1.7.2/
./autogen.sh
export CC=gcc F77=gfortran FC=gfortran LDFLAGS=-lnuma CFLAGS="-g -O3 -march=amdfam15"
./configure --disable-vt --prefix=/opt/openmpi-1.7.2-numaconnect
make
sudo make install
```

Start application like this:

```
mpirun -n 32 -rf rank32 -mca btl self,nc bin/lu.C.32
```

6 IPMI

6.1 Power control

To control the power of a IPMI based node, use the following command(s) :

```
% ipmitool -I lanplus -H <hostname>-ipmi -U <username> -P <password>
power on
% ipmitool -I lanplus -H <hostname>-ipmi -U <username> -P <password>
power off
% ipmitool -I lanplus -H <hostname>-ipmi -U <username> -P <password>
power reset
```

6.2 Console access

To open a serial console session towards an IPMI based node, use the following command :

```
% ipmitool -I lanplus -H <hostname>-ipmi -U <username> -P <password> sol
activate
```

<username> and <password> may be different from device to device.

You can also pipe the console log to a file this way:

```
% ipmitool -I lanplus -H <hostname>-ipmi -U <username> -P <password> sol activate 2>&1 | tee
<filename> &
```

7 INSTALLING APPLICATIONS

7.1 Gromacs

GROMACS

- GROMACS is a versatile package to perform molecular dynamics, i.e. simulate the Newtonian equations of motion for systems with hundreds to millions of particles. It is primarily designed for biochemical molecules like proteins, lipids and nucleic acids that have a lot of complicated bonded interactions, but since GROMACS is extremely fast at calculating the nonbonded interactions (that usually dominate simulations) many groups are also using it for research on non-biological systems, e.g. polymers.
- The NumaConnect Shared Memory test system used to conduct the tests has:
 - 1TB of memory
 - 256 cores
 - It utilizes 8 servers each equipped with:
 - 2 x AMD Opteron 2,5 GHz 6380 CPUs
 - 16 cores in each CPU
 - 128GB of memory

numascale

GROMACS with NC-OpenMPI
[ns/day]
(higher is better)
case: Test-performance_protein-water-membrane.tpr

Number of processes	ns/day
32	19.12
64	24.98
128	39.73

Copyright 2012. All rights reserved.

Gromacs scales positively when using our optimized NumaConnect BTL with OpenMPI:

Core	t (s)	Wall t (s)	%	ns/day	Hour/ns
32	2894.03	90.38	3201.90	19.12	1.26
64	4431.17	69.19	6404.20	24.98	0.96
128	5569.21	43.50	12803.70	39.73	0.60

Make sure you have a suitable FFTW version installed (i.e. single-precision and AMD-optimised). Find some info in [http://www.gromacs.org/Documentation/Installation Instructions for 5.0](http://www.gromacs.org/Documentation/Installation%20Instructions%20for%205.0)
[http://www.gromacs.org/Documentation/Installation Instructions for 5.0#3.4. Fast Fourier Transform library](http://www.gromacs.org/Documentation/Installation%20Instructions%20for%205.0#3.4)
[http://www.gromacs.org/Documentation/Installation Instructions for 5.0#3.2.1. Running in parallel](http://www.gromacs.org/Documentation/Installation%20Instructions%20for%205.0#3.2.1)

Verify the installation by running "make check".

You can easily check your (acceleration) setup with "mdrun -version"

Relevant mdrun options include "-pin on", "-dlb yes", "-tunepme", "-nt [n]" where n is the number of threads. Should be equal to number of cores. If you specify 0, mdrun will guess but it might not guess correctly.

Here are the steps used on the Numascale system:

Compile FFTW:

```
CC=/opt/gcc-4.8.2/bin/gcc CFLAGS="-Inuma" CXX=g++ FC=/opt/gcc-4.8.2/bin/gfortran F77=/opt/gcc-4.8.2/bin/gfortran FFLAGS="-Inuma" ./configure --enable-float --enable-threads --enable-sse2 --prefix=/home/atle/GROMACS/fftw_Inuma
```

Gromacs:

```
CMAKE_PREFIX_PATH=/home/atle/GROMACS/fftw_Inuma cmake .. -DGMX_MPI=ON -DCMAKE_CXX_COMPILER=/opt/openmpi-1.7.2-numaconnect/bin/mpicxx -DCMAKE_C_COMPILER=/opt/openmpi-1.7.2-numaconnect/bin/mpicc -DCMAKE_C_FLAGS="-Inuma" -DCMAKE_CXX_FLAGS="-Inuma" -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/home/atle/GROMACS/MPI -DLIBXML2_LIBRARIES=/usr/lib64 -DGMX_PREFER_STATIC_LIBS=ON -DCMAKE_BUILD_TYPE=Release -DLIBXML2_INCLUDE_DIR=/usr/include/libxml2
```

Currently the MPI version works best on NumaConnect:

```
[atle@numademo mpi_numademo_128]$ mpirun -n 128 -rf rank128_lu -mca btl self,nc /home/atle/GROMACS/MPI_NUMADEMO/bin/mdrun_mpi -s Test-performance_protein-water-membrane.tpr
```