



**THE HIGH-END VIRTUALIZATION COMPANY**  
**SERVER AGGREGATION – CREATING THE POWER OF ONE**



## **Application Execution Guidelines for vSMP Foundation Aggregated Virtual Machine**

Nir Paikowsky – VP Services

Last Update: 06/15/2011

**Aggregate. Scale. Simplify. Save.**

# Background

vSMP Foundation is a virtualization software that creates a single virtual machine over multiple x86-based systems.

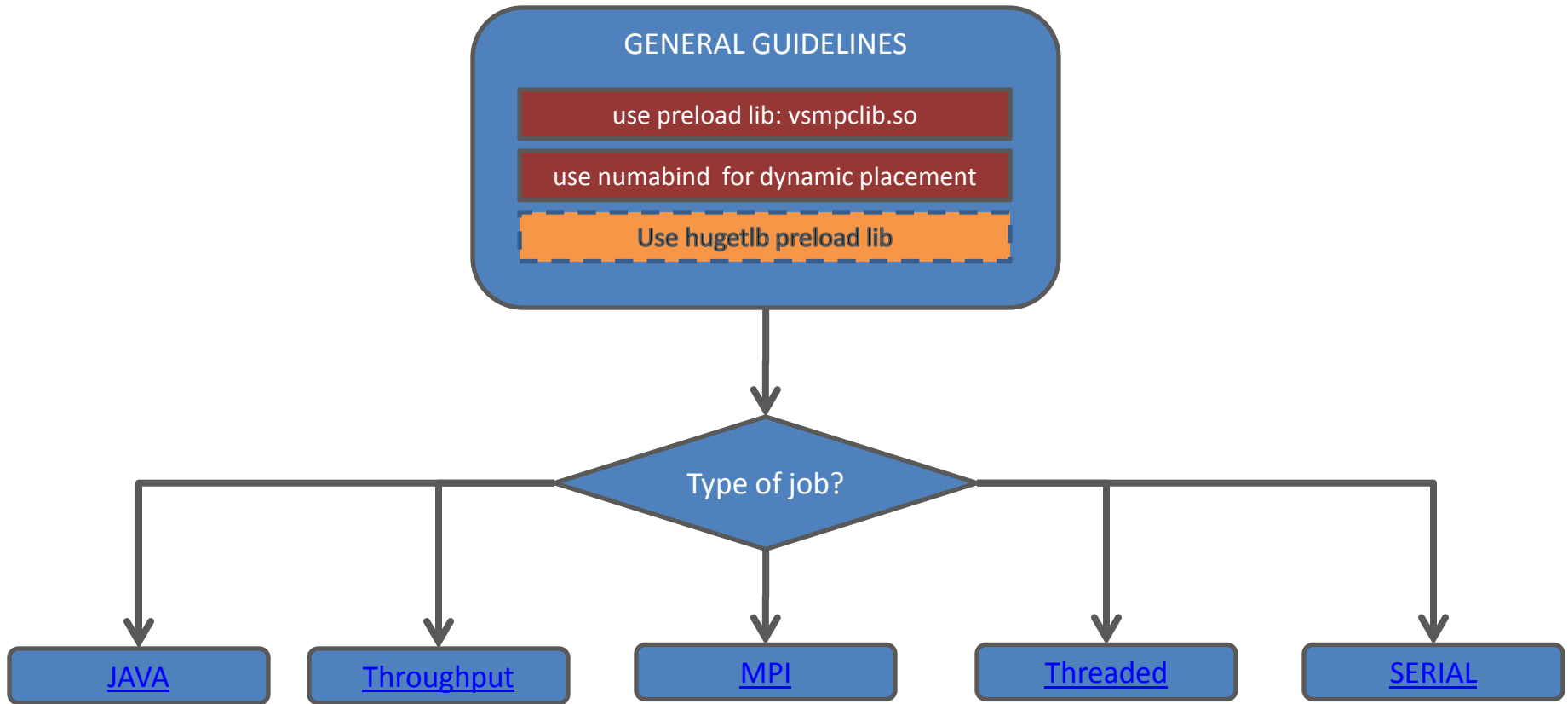
A unique characteristic of vSMP Foundation is its usage of local memory to enhance system performance using sophisticated caching techniques.

Similarly to other large-scale multi-processor systems, processor and system caches can significantly improve application performance if the placement policy for the application's threads or processes (hereafter, “tasks”) is carefully chosen.

# How to use this guide

- **Identify the type of job that best resembles your application** and follow the guidelines and sample run-script (e.g. MPI, OpenMP, etc..)
- **Use in conjunction with the vSMP Productivity Pack (vSMPPP)** which includes:
  - Automatic optimal configuration of your system for HPC workloads
  - Optimized MPI library (MPICH2 tuned for vSMP Foundation)
  - Automatic process placement utility (numabind)
  - Examples and run scripts (see /opt/ScaleMP/examples)
  - And more
- For specific guidelines for common applications please check out support website

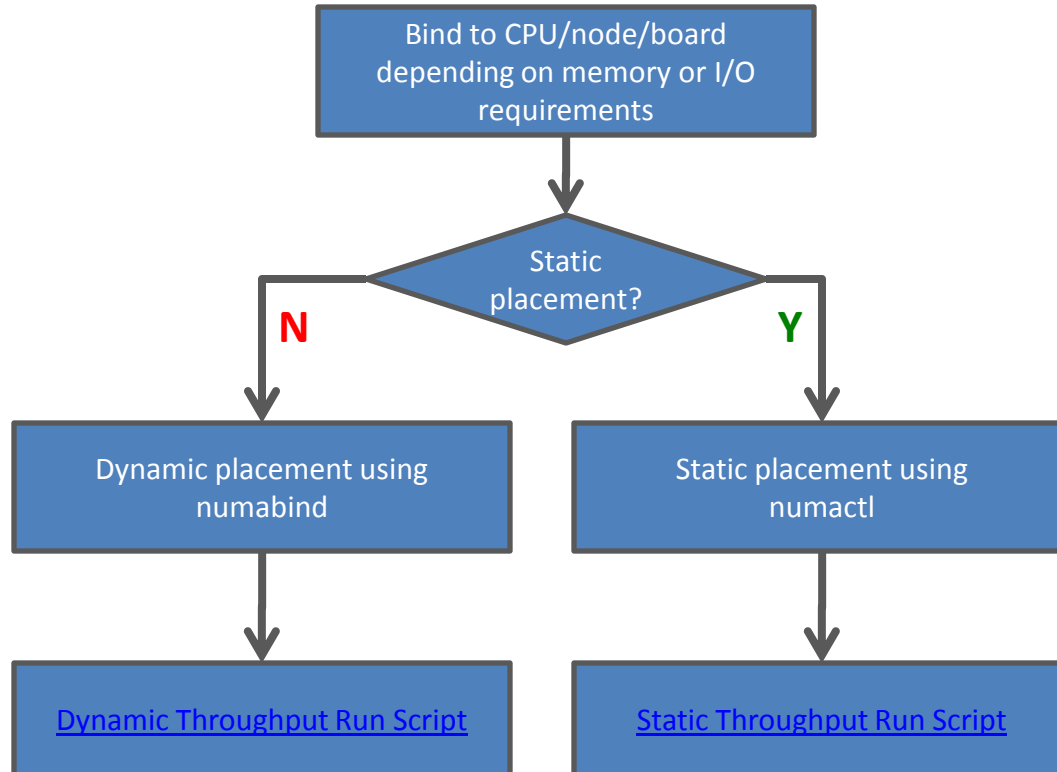
# Application Execution Guidelines



Recommended

Optional

# Serial Job



# Sample Serial Application Run Script

## Example Serial Application (serial-app) run script (static placement)



```
# Setting stacksize to unlimited
ulimit -s unlimited

# ScaleMP preload library that throttles down unnecessary system calls
export LD_PRELOAD=/opt/ScaleMP/libvsmpplib/0.1/lib64/libvsmpplib.so

# Bind the serial job statically to a particular CPU using numactl
cpu=1;

numactl --physcpubind=$cpu ./serial-app > log-serial-app-$cpu.txt
# To bind to a numa node, use numactl -cpunodebind=X

# See /opt/ScaleMP/examples/ for more information
```

# Sample Serial Application Run Script

## Example Serial Application (serial-app) run script (dynamic placement)

```
# Setting stacksize to unlimited
ulimit -s unlimited

# ScaleMP preload library that throttles down unnecessary system calls
export LD_PRELOAD=/opt/ScaleMP/libvsmpclib/0.1/lib64/libvsmpclib.so

# Bind the serial job dynamically to a node/CPU using numabind
./serial-app > log-serial-app-dynamic.txt &

# Wait few seconds for job to be created
sleep 3

# Place the job using numabind
numabind --config myconfig >> log-serial-app-dynamic.txt 2>&1

wait

# See /opt/ScaleMP/examples/ for more information
```



# Sample Serial – numabind config

`name=serial-mm`

`pattern=serial-mm`

`verbose=0`

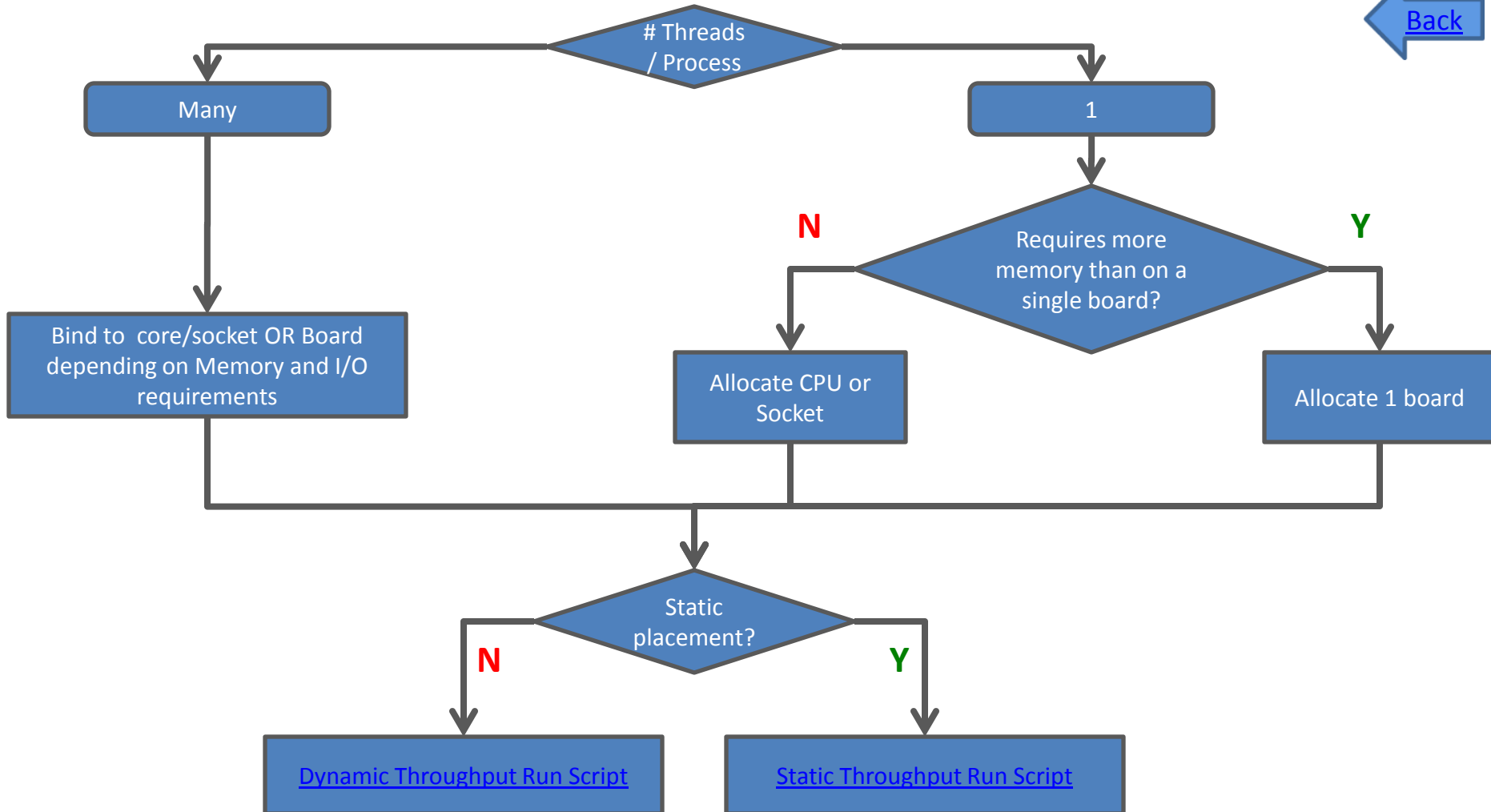
`process_allocation=cpu`

`task_affinity=cpu`

`rule=RULE-throughput.so`



# THROUGHPUT



# Sample Throughput Run Script

Example Throughput Application (throughput-app) run script (static placement)



```
# Setting stacksize to unlimited
ulimit -s unlimited

# ScaleMP preload library that throttles down unnecessary system calls
export LD_PRELOAD=/opt/ScaleMP/libvsmpclib/0.1/lib64/libvsmpclib.so

# Bind the throughput job statically to a particular CPU
for i in 1 2 3 4; do
numactl --physcpubind=$i-1 ./throughput-app $i > log-throughput-app-$i.txt
# To bind to a numa node, use numactl -cpunodebind=X
done

# See /opt/ScaleMP/examples/ for more information
```

# Sample Throughput Run Script

Example Throughput Application (throughput-app) run script (dynamic placement)

```
# Setting stacksize to unlimited
ulimit -s unlimited

# ScaleMP preload library that throttles down unnecessary system calls
export LD_PRELOAD=/opt/ScaleMP/libvsmpclib/0.1/lib64/libvsmpclib.so

# Bind the throughput job dynamically to a node/cpu using numabind
export PATH=/opt/ScaleMP/numabind/bin:${PATH}
for i in 1 2 3 4; do
./throughput-app $i > log-throughput-app-$i.txt &
done

# Wait few seconds for jobs to be created
sleep 5

# Place jobs using numabind
numabind --config myconfig >> log-serial-app-dynamic.txt 2>&1

wait

# See /opt/ScaleMP/examples/ for more information
```



# Sample Throughput – numabind config

```
name=throughput-mm
```

```
pattern=throughput-mm
```

```
verbose=0
```

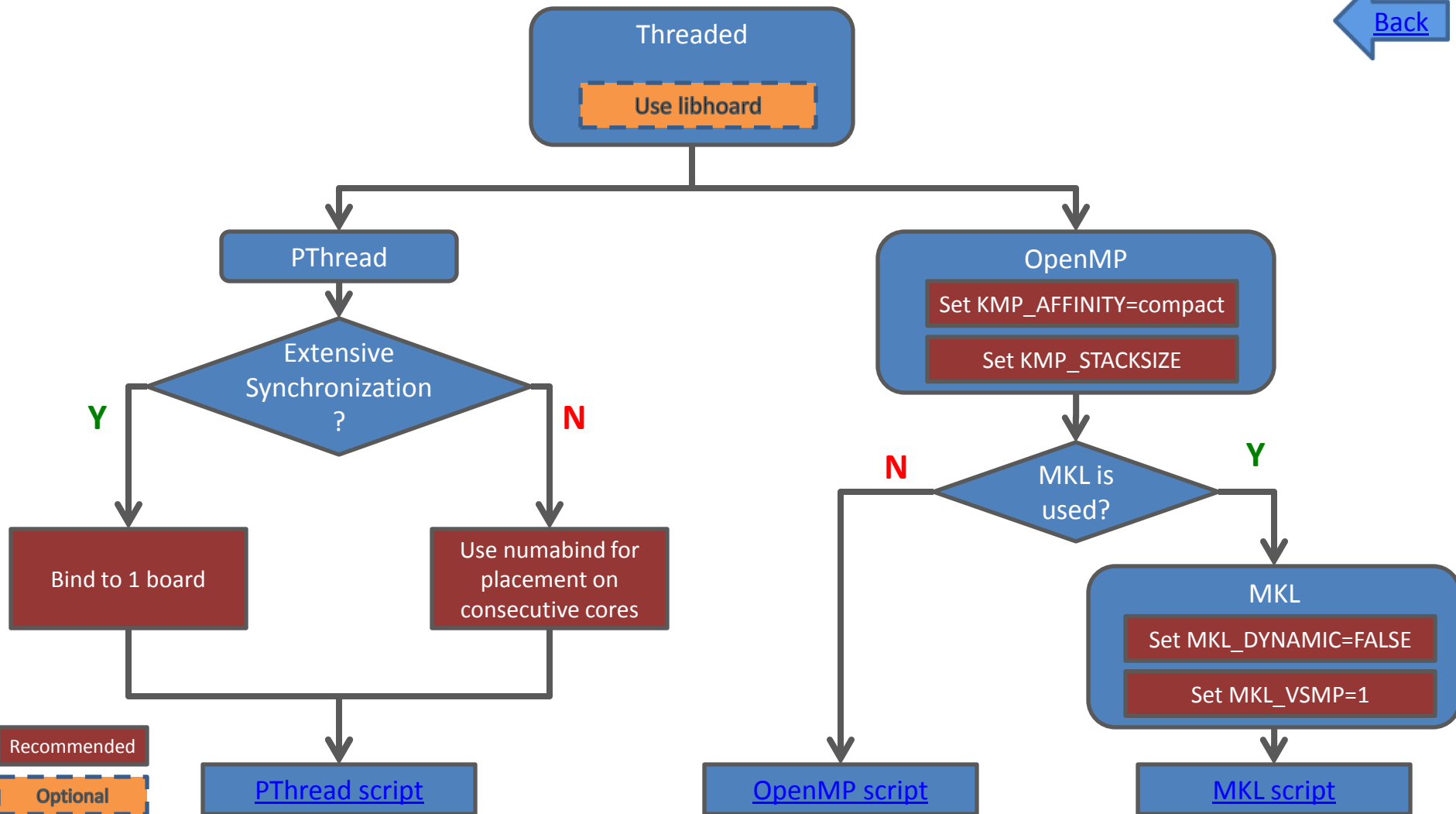
```
process_allocation=cpu
```

```
task_affinity=cpu
```

```
rule=RULE-throughput.so
```



# Threaded Jobs



# Sample OpenMP Run Script

## Example OpenMP Application (openmp-app) run script

```
# Make sure to use Intel compilers to build the application
# Intel compiler runtime environemnt setting
source /opt/intel/Compiler/11.1/069/bin/iccvars.sh intel64
```

```
# Setting stacksize to unlimited
ulimit -s unlimited
```

```
# ScaleMP preload library that throttles down unnecessary system calls
export LD_PRELOAD=/opt/ScaleMP/libvsmpplib/0.1/lib64/libvsmpplib.so
```

```
# Bind OpenMP threads statically to consecutive cpus: last number is the cpu to start from
export KMP_AFFINITY=compact,verbose,0,0
```

OR

```
# Alternatively you can use dynamic placement using numabind
# export PATH=/opt/ScaleMP/numabind/bin:${PATH}
# export LD_LIBRARY_PATH=/opt/ScaleMP/numabind/lib
# The cpu/offset to start from is calculated dynamically by numabind
# export KMP_AFFINITY=compact,verbose,0,`numabind --offset $np`
```

```
np=16
export OMP_NUM_THREADS=$np
/usr/bin/time ./openmp-app > log-openmp-app-$np.txt
```

```
# See /opt/ScaleMP/examples/OpenMP for more information
```



# Sample MKL Run Script

## Example MKL Application (mkl-app) run script

```
# Make sure to use Intel compilers to build the application
# Intel compiler runtime environemnt setting
source /opt/intel/Compiler/11.1/069/bin/iccvars.sh intel64
```

```
# Setting stacksize to unlimited
ulimit -s unlimited
```

```
# ScaleMP preload library that throttles down unnecessary system calls
export LD_PRELOAD=/opt/ScaleMP/libvsmpplib/0.1/lib64/libvsmpplib.so
```

```
# Make sure to set this variable to FALSE
export MKL_DYNAMIC=FALSE
export MKL_VSMP=1
```

```
# Bind OpenMP threads statically to consecutive cpus: last number is the cpu to start from
export KMP_AFFINITY=compact,verbose,0,0
```

OR

```
# Alternatively you can use dynamic placement using numabind
# export PATH=/opt/ScaleMP/numabind/bin:${PATH}
# export LD_LIBRARY_PATH=/opt/ScaleMP/numabind/lib
# The cpu/offset to start from is calculated dynamically by numabind
# export KMP_AFFINITY=compact,verbose,0,`numabind --offset $np`
```

```
np=16
export OMP_NUM_THREADS=$np
/usr/bin/time ./openmp-app > log-mkl-app-$np.txt
```

```
# See /opt/ScaleMP/examples/OpenMP for more information
```



# Sample PThread Run Script

## Example PThread Application (pthread-app) run script

```
# ScaleMP preload library that throttles down unnecessary system calls
export LD_PRELOAD=/opt/ScaleMP/libvsmpclib/0.1/lib64/libvsmpclib.so

# PATH to numabind, See /opt/ScaleMP/examples/ for more information
export PATH=/opt/ScaleMP/numabind/bin:$PATH

# Specify sleep duration for each pthread. Default = 60 sec if not set
export SLEEP_TIME=120

# 16 pthreads would be created
NP=16

./ptest $NP > log-pthread-app-$NP 2>&1 &

# Waiting for 15 seconds for all the threads to start
sleep 15

# Start numabind with a config file that has a rule for pthread,
# which would place all threads to consecutive cpus.
numabind --config myconfig >> log-pthread-app-$NP 2>&1

wait

# See /opt/ScaleMP/examples/Pthread for more information
```



# Sample PThread – numabind config

`name=ptest`

`pattern=ptest`

`verbose=0`

`process_allocation=multi`

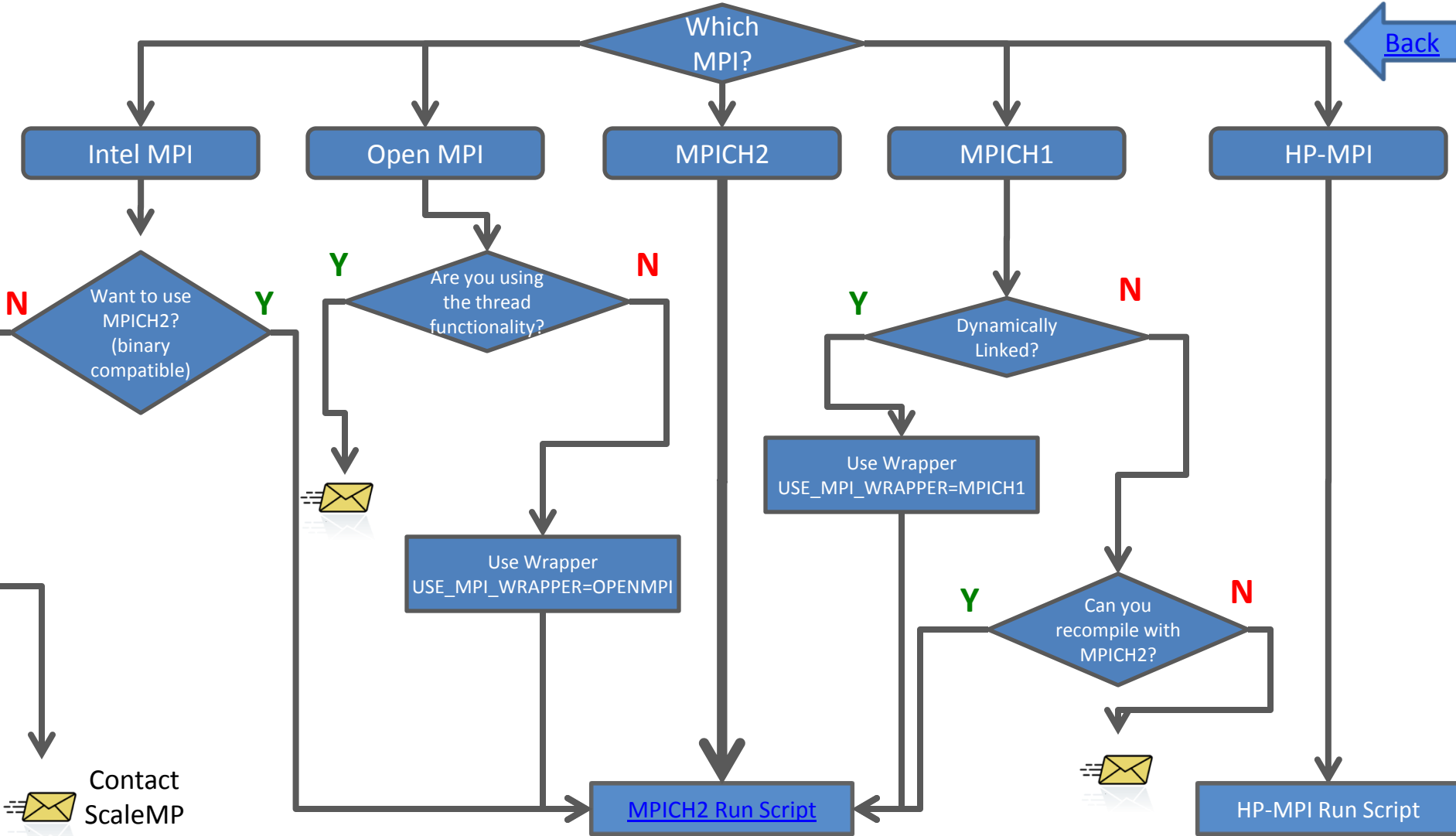
`task_affinity=cpu`

`rule=RULE-procgroup.so`

`flags=ignore_idle`



# MPI



Aggregate. Scale. Simplify. Save.

# Sample MPICH2 Run Script

## Example MPICH2 Application (mpi-app) run script



```
ulimit -s unlimited
```

```
# Add path to MPICH2 tuned for vSMP foundation
```

```
source ../mpich.sh
```

```
# If application binary is MPICH1 and dynamically linked, uncomment this
```

```
# export USE_MPI_WRAPPER=MPICH1
```

```
# ScaleMP preload library that throttles down unnecessary system calls
```

```
export LD_PRELOAD=/opt/ScaleMP/libvsmpclib/0.1/lib64/libvsmpclib.so
```

```
# Bind MPI processes on consecutive cores
```

```
export VSMP_PLACEMENT=PACKED
```

```
# Adding numabind for dynamic placement (according to VSMP_PLACEMENT)
```

```
export PATH=/opt/ScaleMP/numabind/bin:${PATH}
```

```
export VSMP_VERBOSE=YES
```

```
export VSMP_MEM_PIN=YES
```

```
export ncpus=`grep -w processor /proc/cpuinfo | wc -l`
```

```
time mpirun -np $NPROC ./mpi-app > log-mpi-app-$NPROC.txt
```

```
# See /opt/ScaleMP/examples/ for more information
```

# How to Contact ScaleMP



Open a support ticket through our support portal:

<http://support.scalemp.com>

Or send an email to:

[support@scalemp.com](mailto:support@scalemp.com)

# List of Examples

/opt/ScaleMP/examples/

- **Serial/**
  - runserial: Basic serial job using numactl for static placement
  - runserial-nbind: Same as above using numabind for dynamic placement
- **Throughput/**
  - runthroughput: Basic throughput job using 4 processes
  - Runthroughput-nbind: Same as above using numabind for dynamic placement
- **OpenMP/**
  - runopenmp : Basic OpenMP example with static placement
  - runopenmp -nbind: OpenMP example using numabind for dynamic placement
  - runopenmp2: Running 2 OpenMP jobs concurrently using numabind
- **MKL/**
  - runmkl: Running Intel MKL job
  - runmkl-nbind: Same above with dynamic placement
- **Pthread/**
  - run: Running 16 thread using pthreads with placement by numabind on consecutive cores
- **MPICH2/**
  - runmpi: Basic MPI example using MPICH2 tuned for vSMP Foundation with static placement
  - runmpi-nbind: Same as above using numabind for dynamic placement
  - runmpis: Running 2 MPI jobs concurrently using numabind
- **MPICH1/ and OpenMPI/**
  - Same as MPICH2 example, but using the MPICH1 wrapper or OpenMPI wrapper
  - The 2 wrappers do not include implementation of all functions yet, but the most common ones. Source is available for customers to add additional functions implementations



**THE HIGH-END VIRTUALIZATION COMPANY**  
**SERVER AGGREGATION – CREATING THE POWER OF ONE**



info@ScaleMP.com, +1 (408) 342 0330

**Aggregate. Scale. Simplify. Save.**