Empowered by Innovation NEC

NEC SX vectorization workshop

Jens-Olaf Beismann
Senior Benchmarking Analyst
NEC Deutschland GmbH

Objectives …

```
      ******  Program Information  ******
Real  Time (sec)          :       274.312165
User  Time (sec)          :       272.141663
System Time (sec)         :         1.473929
Vector Time (sec)         :       257.934811
Instruction Count         :     38211119460
Vector Instruction Count  :      8688304757
Vector Element Count      :   2057078964016
FLOP Count                :    890893604266
MOPS                      :      7667.336768
MFLOPS                    :      3273.639157
Average Vector Length     :       236.764135
Vector Op. Ratio (%)      :        98.585125
Memory size used (MB)     :      1497.01942
MIPS                      :       140.408929
Inst. Cache miss (sec)    :         0.229672
Operand  Cache miss (sec):          2.678918
Bank  Conflict Time (sec):          9.299637

 Start Time (date)  :  2004/08/12 05:25:08
 End   Time (date)  :  2004/08/12 05:29:46
```


Agenda

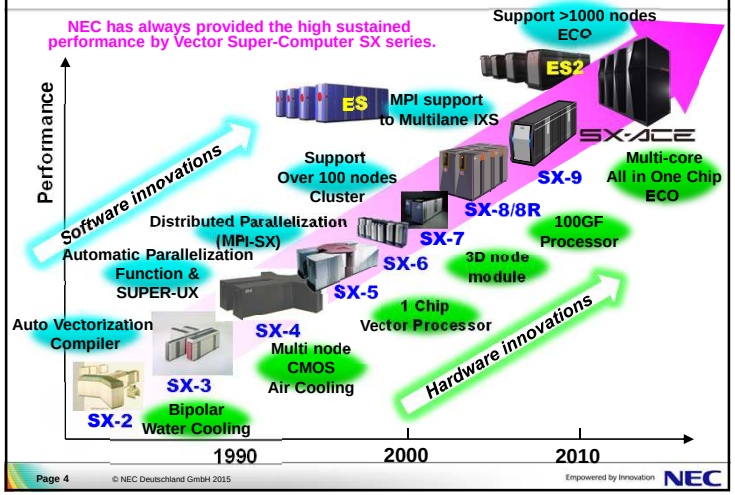
DAY 1

9:30 – 11:00 Overview and basics on vectorization
11:00 – 11:30 Coffee break
11:30 – 13:00 Compilers, options and performance analysis
13:00 – 14:00 Lunch break
14:00 – 15:30 Optimization and vectorization examples, indirect addressing
16:00 – 17:30 Run-time environment, Parallelization on SX

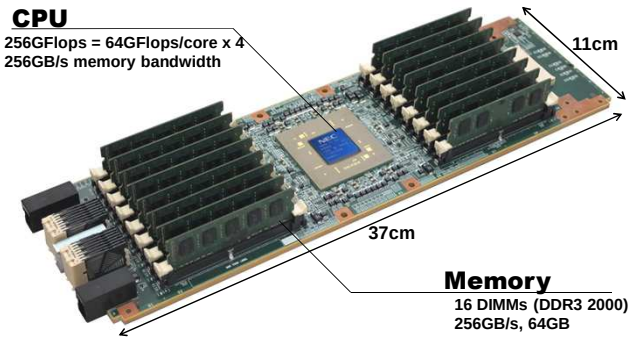
DAY 2

9:00 – 10:30 NEC SX-ACE: Architecture and new features
10:30 – 11:00 Coffee break
11:00 – 12:30 SX-ACE usage at HLRS
12:30 – 13:30 Lunch break
13:30 – 16:30 Hands-on session

SX History and Technical Evolutions



SX-ACE node card



Rated power consumption = 469W

SX-ACE : 2-node module

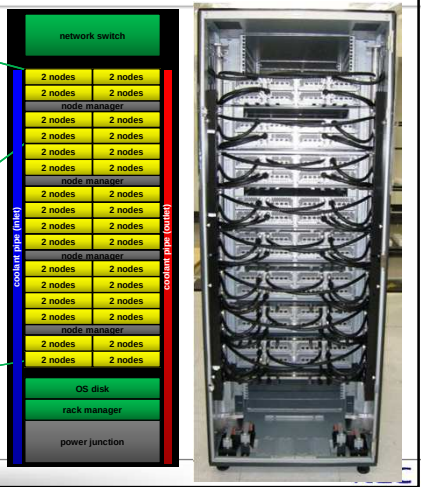
2-node module
2 node-cards are implemented
CPUs are cooled by water flow

16-node cage
8 2-node modules are implemented

SX-ACE : Rack implementation

16-node cage

2-node module

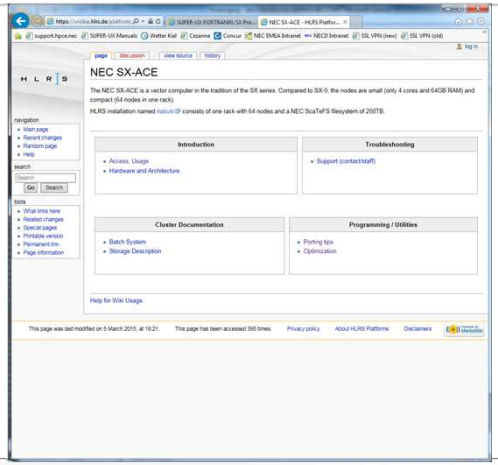


Empowered by Innovation NEC

SX Documentation

Available at `http://fs.hlrs.de/~hwwnec5/SX-ACE`

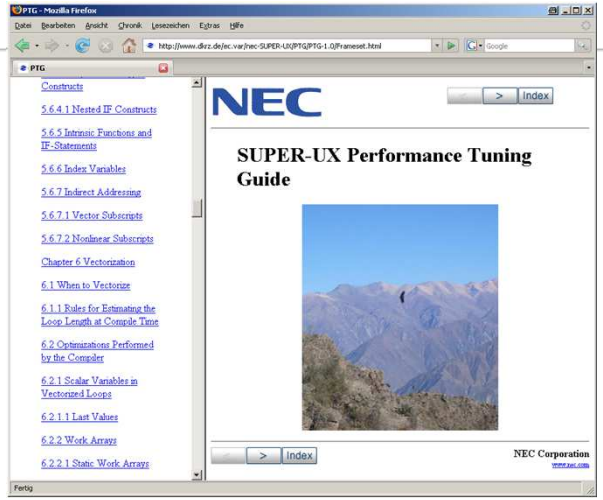
SX Documentation



Programming languages

- **NEC**
- FORTRAN90/SX Programmer's Guide
- FORTRAN90/SX Language Reference Manual
- C++/SX Programmer's Guide

- **Vectorization**
- SUPER-UX Performance Tuning Guide
- FORTRAN90/SX Programmer's Guide



Empowered by Innovation NEC

Hardware Aspects of Vector Computing

How do Vector Computers work?

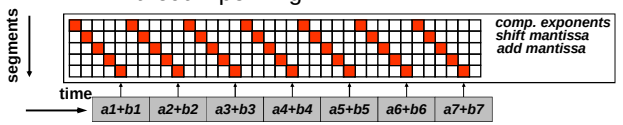
Operation Segmentation:

- Operations are decomposed into segments
- Example: addition of floating point number
  - compare exponent
  - shift mantissa
  - add mantissa
  - select exponent and normalize

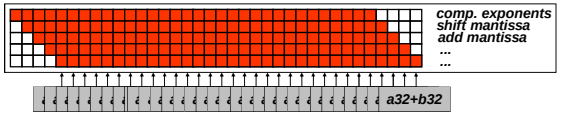
```
1.14e9 - 2.78e8
1.14e9 - 0.278e9
0.862e9
8.62e8
```


How do Vector Computers work?

- Pipelining:
  - without Pipelining

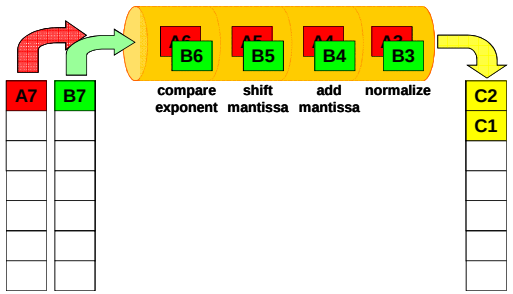


  - with Pipelining

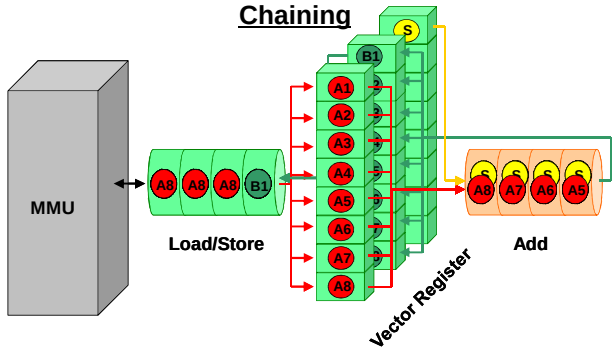


How do Vector Computers work?

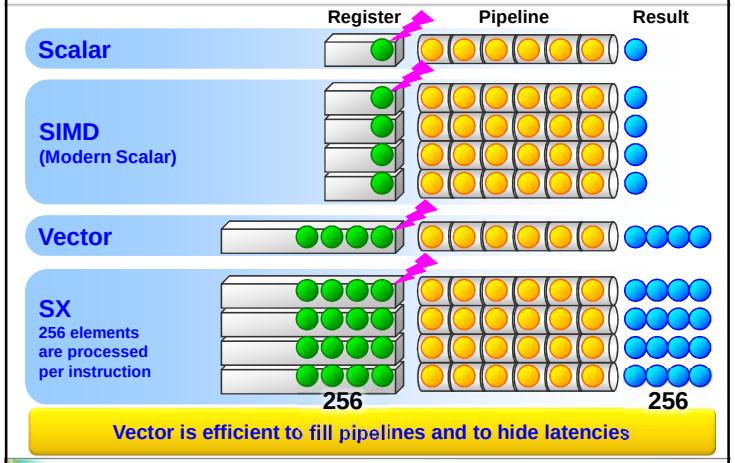
- **Vector "Add" Pipeline:**



How do Vector Computers work?



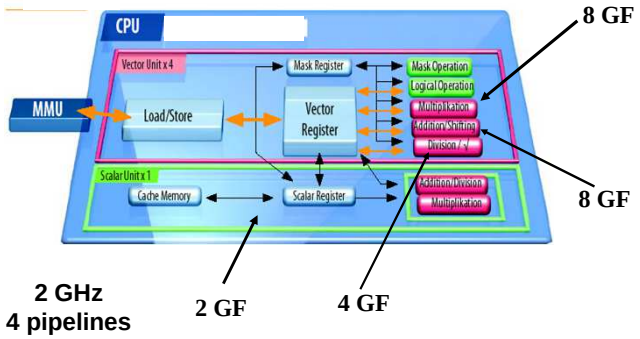
SIMD = Vector ?



Empowered by Innovation NEC

SX processor architecture

SX-8: 16 GFLOPs Peak and 64 GB/s bandwidth



Data Parallelism

"Vector Loop": Data Parallel

```
do i = 1, n
  a(i) = b(i) + c(i)
end do
```

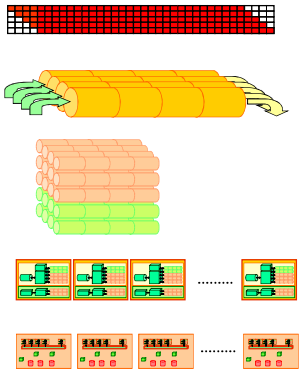
```
for (i=0; i<n; i++)
   a[i] = b[i] + c[i];
```

"Scalar Loop", example: recursion

```
do i = 2, n
  a(i) = a(i-1) + b(i)
end do
```

Levels of Parallelism

- Segmentation
- multiple pipes
- parallel usage of functional units
- parallel CPUs
- parallel nodes



Levels of Parallelism

- Segmentation
- multiple pipes
- parallel usage of functional units
- parallel CPUs
- parallel nodes

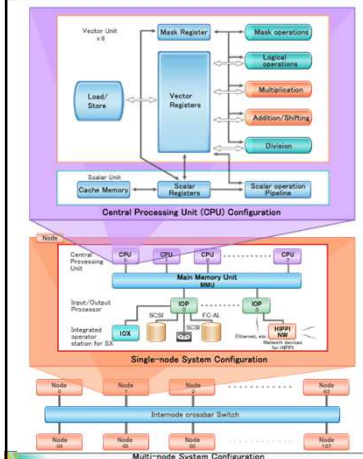
Efficiency:

$$\frac{\text{used units}}{\text{available units}}$$

To achieve high efficiency:

- **Keep all units busy**

Parallel programming on SX architecture



CPU level
- **Vectorisation**

Shared Memory level
- **Microtasking**
- **Macrotasking**
- **Autotasking**
- **OpenMP**
- **MPI**

Hybrid

Distributed Memory level
- **MPI**
- **Co-array Fortran**

Thinking Vector ?

- ‘Scalar’ thinking / coding:

There is a (grid-point,particle,equation,element), what am I going to do with it?

- ‘Vector’ thinking / coding:

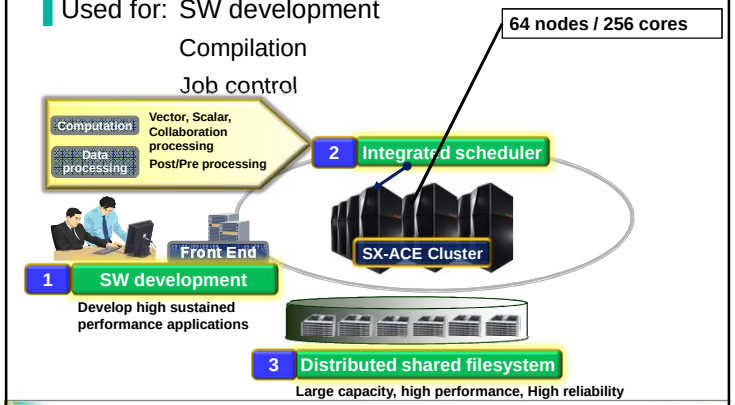
There is a certain action or operation, to which (grid-points,particles,equations,elements) am I going to apply it simultaneously?

Empowered by Innovation NEC

First steps: compiling a code, submitting a job

System overview

- Login point: kabuki (1)
- Used for: SW development
  Compilation
  Job control



Basic steps on the SX

Don‘t use the SX for compiling and linking !
Use cross compilers on frontend (**`kabuki`**)

`sxc++`	C++ and C Compiler
`sxf90`	F90 Compiler
`sxar`	ar archiver
`sxmpic++`	MPI C++/C Compiler
`sxmpif90`	MPI F90 Compiler
`sxftrace`	Analyzing Trace files

FORTRAN90/SX

Syntax :

```
sxf90  <command line options> \
       -Wf,<detailed optimisation options> \
       -Wf,<detailed vectorisation/parallelisation options> \
       -Wf,<miscellaneous options> \
       -Wl,<linker options> …
```

Basic set of compiler options

`-R2`	transformation and formatted listing
`-Wf,-L summary`	summary of compiler options
`-pi`	enable automatic inlining
`-Chopt`	highest optimisation level
`-Wf,-pvctl fullmsg`	full vectorisation diagnostics
`-Wf,-pvctl noassume loopcnt=1000000`	information about loop lengths

Command line options : **optimization**

F90 and C++

-C aopt	(only C++) agressive vectorization and optimization
-C hopt	Full use of vectorization and optimization
-C vopt	Standard use of vectorization and optimization
-C sopt	Full use of optimization in scalar code
-C vsafe	Very safe use of vectorization and optimization
-C ssafe	Safe use of optimization in scalar code
-C noopt	(only C++) almost all optimizations and vectorization are suppressed
-C debug	No vectorization and optimization at all

Useful environment variables

F_PROGINF=[NO|YES|DETAIL]
performance information from hardware counters

F_SETBUF=n
set I/O buffers to n KB

F_FTRACE=[NO|FMT1|FMT2]
flow trace performance information

F_FILEINF=[NO|YES|DETAIL]
I/O statistics

Most environment variables also available as `C_<varname>` !

How to compile an MPI code

```
sxf90 code.f90 –o code

sxmpif90 mpicode.f90 –o mpicode
```

sxmpif90 does it all for you – no need to link or include anything !

NQSII Batch Example Single Node

```
#!/usr/bin/ksh
#PBS -q smallque                        # queue name
#PBS -b 1                               # number of nodes
#PBS -l cpunum_job=4                    # cpus per Node
#PBS -l elapstim_req=02:00:00           # max wallclock time
#
#PBS -l memsz_job=10gb                  # max accumulated memory
#PBS -j o                               # join stdout/stderr
#PBS -N MyJob                           # job name
#

cd $WORK
cp $HOME/code/myexec .

export MPIPROGINF=YES                   # give some performance information

mpirun -np 4 ./myexec                   # start the code with 4 mpi processes
cp outfile $HOME/code
                                        # please be so kind to clean your scratch
                                        # space if files are not needed any more!
rm *.data                               # ... or whatever your files are called
```

qsub script

NQSII Batch Example 2 Nodes

```
#!/usr/bin/ksh
#PBS -q <queue name>
#PBS -l cpunum_job=4                    # cpus per Node
#PBS -b 2                               # number of nodes
#PBS -T mpisx
#PBS -l elapstim_req=02:00:00           # max wallclock time
#
#PBS -l memsz_job=10gb                  # max accumulated memory
#PBS -j o                               # join stdout/stderr
#PBS -N MyJob                           # job name
#

cd <somewhere>
cp $HOME/code/myexec .

export MPIPROGINF=YES                   # give some performance information

export MPIEXPORT="MPIPROGINF"

mpirun -nn 2 -nnp 4 ./myexec            # start the code with 4 mpi processes per node

cp outfile $HOME/code
```

qsub script

NQSII Batch Job Monitoring

qstat -Q – which queues ?

```
necbench@fe00:~> qstat -Q
[EXECUTION QUEUE] Batch Server Host: siox
=========================================
QueueName       SCH JSVs ENA STS  PRI  TOT ARR WAI QUE PRR RUN POR EXT HLD HOL RST SUS MIG STG CHK
--------------- --- ---- --- --- ---- ---- --- --- --- --- --- --- --- --- --- --- --- --- --- ---
express           1    2 ENA ACT   21    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
front             1    0 ENA ACT   30    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
multi             1    4 ENA ACT   30    1   0   0   0   0   1   0   0   0   0   0   0   0   0   0
standard          1    4 ENA ACT   20    1   0   0   0   0   1   0   0   0   0   0   0   0   0   0
--------------- --- ---- --- --- ---- ---- --- --- --- --- --- --- --- --- --- --- --- --- --- ---
 <TOTAL>                                  2   0   0   0   0   2   0   0   0   0   0   0   0   0

[ROUTING QUEUE] Batch Server Host: siox
=======================================
QueueName       ENA STS PRI RLM  TOT ARR WAI QUE HLD TRS
--------------- --- --- --- ---- --- --- --- --- --- ---
batch           ENA ACT  20  15    0
--------------- --- --- --- ---- --- --- --- --- --- ---
 <TOTAL>                          100   0
--------------- --- --- --- ---- --- --- --- --- --- ---

[NETWORK QUEUE] Batch Server Host: siox
=======================================
QueueName       StagingMachine      ENA STS PRI  RLM  TOT WAI QUE RUN
--------------- ------------------- --- --- ---- ---- ---- --- --- ---
DefaultNetQue    (any)              ENA ACT   0    1    0   0   0   0
--------------- ------------------- --- --- ---- ---- ---- --- --- ---
 <TOTAL>                                        1000   0   0   0   0
--------------- ------------------- --- --- ---- ---- ---- --- --- ---

necbench@fe00:~>
```

qstat -Qf <que> – what limits ?

NQSII Batch Job Monitoring

qstat -Qf <que> – what limits ?

```
Resources Limits:
  (Per-Req) Elapse Time Limit       = Max:     252000 Warn:     252000 Std:     252000
  (Per-Job) CPU Time                = Max:      86400 Warn:      86400 Std:      86400
  (Per-Job) CPU Resident Number     = Max: UNLIMITED Warn:        --- Std: UNLIMITED
  (Per-Job) CPU Resident Time       = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  (Per-Job) CPU Number              = Max: UNLIMITED Warn:        --- Std: UNLIMITED
  (Per-Job) Tape Number             = Max: UNLIMITED Warn:        --- Std: UNLIMITED
  (Per-Job) Open File Number        = Max: UNLIMITED Warn:        --- Std: UNLIMITED
  (Per-Job) Process Number          = Max: UNLIMITED Warn:        --- Std: UNLIMITED
  (Per-Job) Memory Size             = Max:        45G Warn:        45G Std:        26G
  (Per-Job) Permanent File Capacity = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  (Per-Job) Temporary File Capacity = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  (Per-Job) RLG 0 File Capacity     = Max: UNLIMITED Warn: UNLIMITED Std:        4M
  (Per-Job) RLG 1 File Capacity     = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  (Per-Job) RLG 2 File Capacity     = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  (Per-Job) RLG 3 File Capacity     = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  (Per-Job) GPID Number             = Max: UNLIMITED Warn:        --- Std: UNLIMITED
  (Per-Prc) CPU Time                = Max:      86400 Warn:      86400 Std:      86400
  (Per-Prc) CPU Resident Number     = Max: UNLIMITED Warn:        --- Std: UNLIMITED
  (Per-Prc) CPU Resident Time       = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  (Per-Prc) CPU Number              = Max:         6 Warn:        --- Std:         2
  (Per-Prc) Task Number             = Max: UNLIMITED Warn:        --- Std: UNLIMITED
  (Per-Prc) Open File Number        = Max: UNLIMITED Warn:        --- Std: UNLIMITED
  (Per-Prc) Memory Size             = Max:        45G Warn:        45G Std:        26G
  (Per-Prc) Data Segment Size       = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  (Per-Prc) Stack Segment Size      = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  (Per-Prc) Core File Size          = Max: UNLIMITED Warn: UNLIMITED Std:         0B
  (Per-Prc) Permanent File Size     = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  (Per-Prc) Permanent File Capacity = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
```


NQSII : manipulating jobs, queue information, …

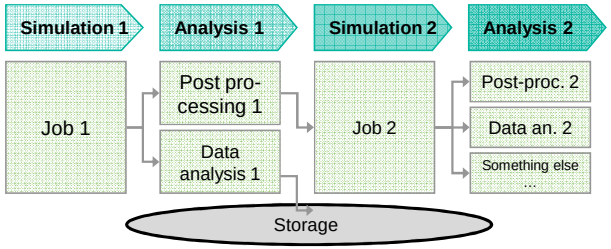
On `kabuki` :

Some useful NQS**`II`** commands :

- **`qdel 1726`** — delete job with job ID 1726
- **`qstat –Qf smallque`** — get full information about settings of queue “smallque”
- **`qcat –o 1726`** — see job output (log file contents) during execution
- **`man qsub`** — access man pages for NQSII qsub command

Job workflow

Arranging a set of jobs with dependencies…



NQSII workflow tools.

NQSII workflow tools

- NQSII commands **`wstart, wdel, wstat`**
  `qsub` (and options)
  `qwait, qwait2`
- Describe job workflow in a “workflow script”

```
#!/bin/bash

qsub -N TEST32n-1 ORCA025.L46-KBM01_1node.ksh
qsub -N TEST32n-2 ORCA025.L46-KBM01_1node.ksh --after TEST32n-1
qsub -N TEST32n-3 ORCA025.L46-KBM01_1node.ksh --after TEST32n-2

qwait2 TEST32n-3
```


NQSII workflow tools

```
nesh-fe : wstart wfl.sh >log.wfl 2>&1 &

nesh-fe : wstat
WFL-ID       Request Owner    Stat
------------ ------- -------- ----
2_ace-ssiox        3 xeext098 RUN

nesh-fe : wstat 2
WFL-ID       RequestID    RequestName  Stat Exit
------------ ------------ ------------ ---- ----
2_ace-ssiox  78436.ace-ss TEST32n-1    RUN     -
2_ace-ssiox  78437.ace-ss TEST32n-2    QUE     -
2_ace-ssiox  78438.ace-ss TEST32n-3    QUE     -

nesh-fe : qstat
RequestID       ReqName  UserName Queue    Pri STT S   Memory      CPU   Elapse R H M Jobs
--------------- -------- -------- -------- ---- --- - -------- -------- -------- - - - ----
78436.ace-ssiox TEST32n- xeext098 smallque    0 RUN -    1.03G     9.75       65 Y Y Y   32
78437.ace-ssiox TEST32n- xeext098 smallque    0 QUE -    0.00B     0.00        0 Y Y Y   32
78438.ace-ssiox TEST32n- xeext098 smallque    0 QUE -    0.00B     0.00        0 Y Y Y   32
```


NQSII workflow tools

Simple cases :

```
kabuki : qsub job1 job2 job3 job4 ... jobN

       → job chain (N consecutive jobs)

kabuki : qsub job1:job2:job3:job4:...:jobN

       → N parallel jobs running

kabuki : qsub --after <nqs-id> job1

       →  start after job <nqs-id> has finished
```


Empowered by Innovation

NEC