Empowered by Innovation NEC

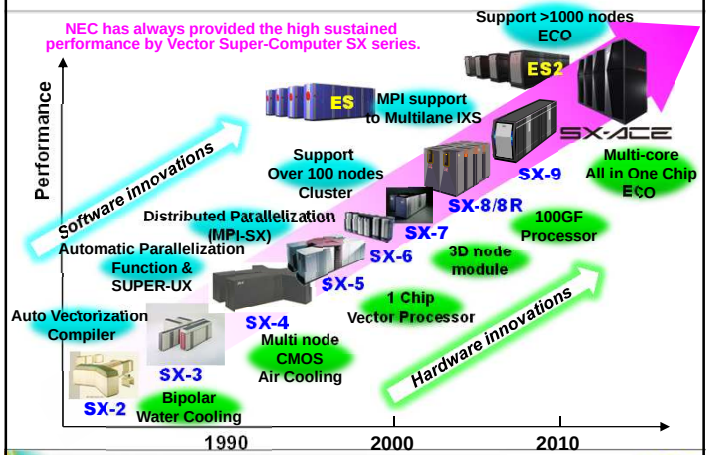
NEC SX-ACE

Jens-Olaf Beismann
Senior Benchmarking Analyst
NEC Deutschland GmbH

SX-ACE Introduction

- **Processor evolution**
- **SX-ACE architecture**
  - Design concept
  - Hardware implementation
- **Performance**
  - Analysis
  - Optimization
- **Usage aspects**
  - NQSII batch system
  - Scalable Technology File System

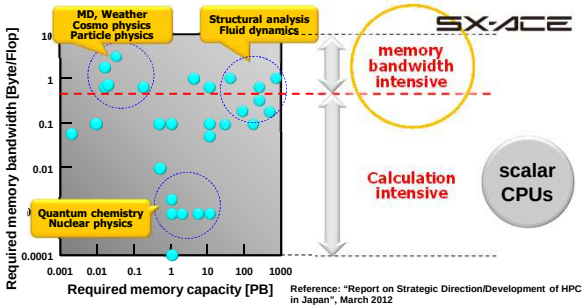
SX History and Technical Evolutions



Code balance : Byte/Flop requirements in real applications

According to Japanese Government (MEXT) working group report of wide variety of strategic segment applications, diverse characteristics are observed.

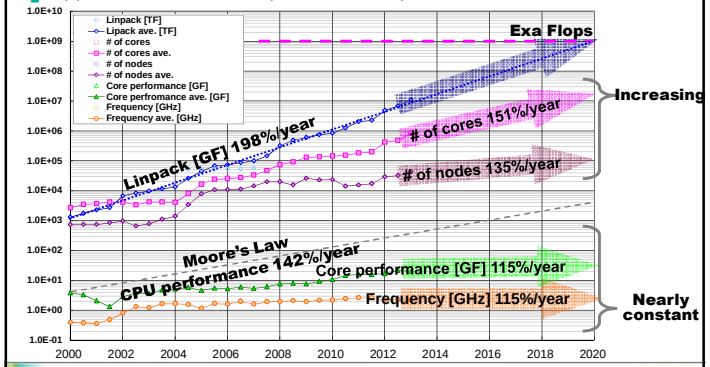
MEXT: Ministry of Education, Culture, Sports, Science & Technology



Reference: "Report on Strategic Direction/Development of HPC in Japan", March 2012

TOP500 performance trend (top 10 systems)

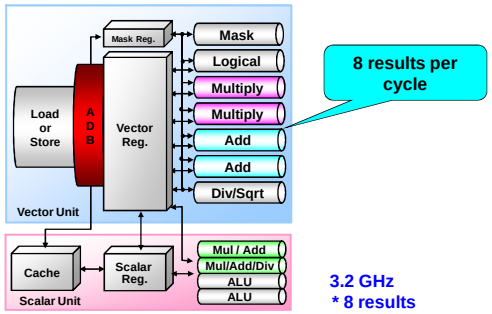
- Increase in LINPACK performance due to higher #cores, #nodes
- Applications have to exploit massive parallelism



Empowered by Innovation NEC

SX-ACE processor architecture

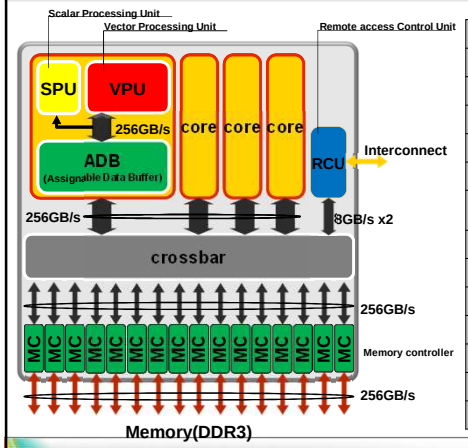
Reminder : SX-9



8 results per cycle

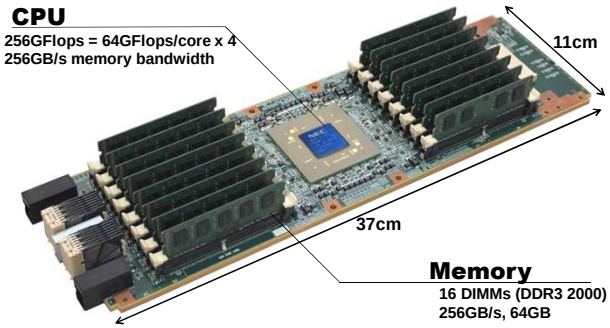
3.2 GHz
* 8 results
* (2 add + 2 mult)
→ 102.4 GFlops

SX-ACE



Architecture	Vector
CORE	
Clock Frequency	1.0GHz
Performance	64GFlops
ADB size	1MB
ADB bandwidth	256GB/s
Memory bandwidth	64GB/s~ 256GB/s
Memory Byte/Flop	1.0 ~ 4.0
CPU	
Cores	4
Performance	256GFlops
Memory bandwidth	256GB/s
Byte/Flop	1.0
Memory capacity	64GB

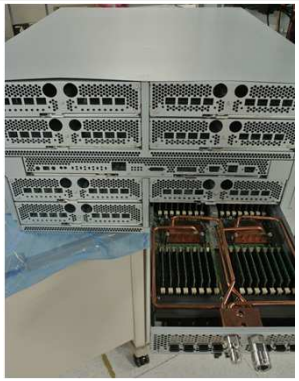
SX-ACE node card



Rated power consumption = 469W

SX-ACE : 2-node module

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

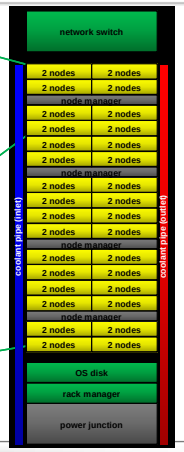


16-node cage
8 2-node modules are implemented

SX-ACE : Rack implementation

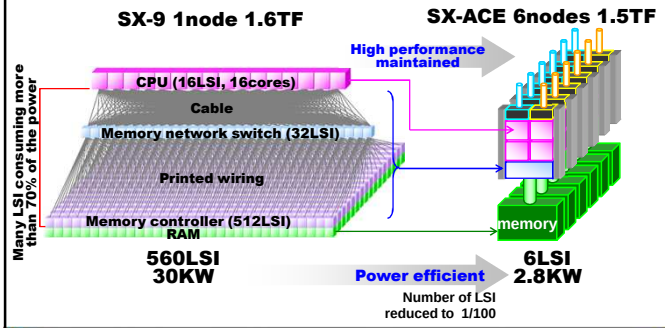
16-node cage

2-node module



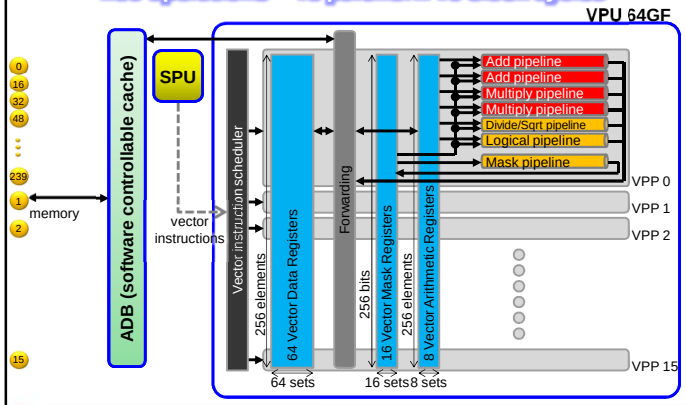
Memory Network Integration

- Large SMP configuration can provide high sustained performance
- But, over 70% power was consumed by the memory network
- SX-ACE processor integrates the memory network into LSI



Core Architecture

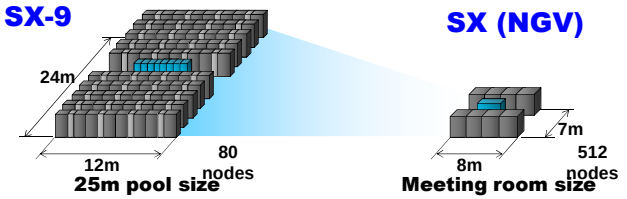
256 operations = 16 parallel x 16 clock cycles



Downsizing and Power Saving

Providing 5x smaller space and 10x lower power consumption compared to SX9 by power saving design and compact implementation.

Comparison with same performance (131TF)



SX-9		SX (NGV)
131TF		131TF
288m²	space 1/5	56m²
2.4MW	power 1/10	0.24MW

Concepts of SX-ACE

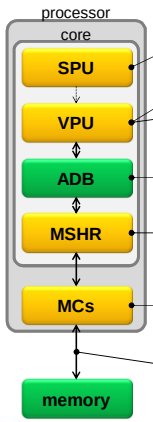
The best solution for memory intensive APs against scalar processors trend

Big Core
Reducing Massive Parallel Difficulty with fewer cores
The highest performance: 64GF
The largest memory bandwidth: 64~256GB/s

Low Power Consumption
The best memory bandwidth solution
GB/s / Watt compared x86 CPU 1.8x

Hybrid Solution
Vector / Scalar tightly coupled environment
Specialized SWs

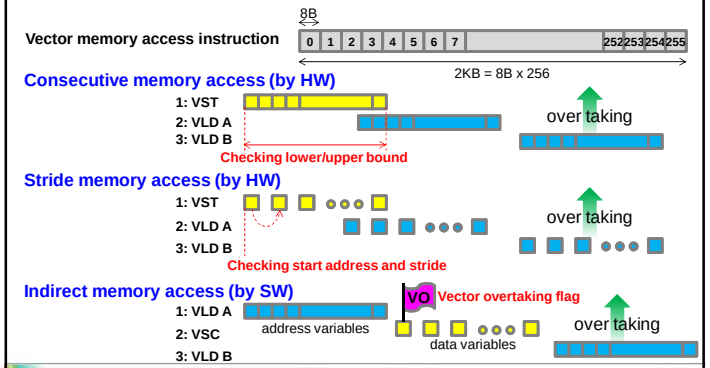
Memory Subsystem and Functions



- Vector instruction (256e) is issued from SPU to VPU
- OoO vector memory access by HW/SW
- **Load request compression (Traffic Reduction)**
  requests (1e each) to same cache line are merged
- **SW controllable cache (Traffic Reduction)**
  compiler/user can control data assign to avoid pollution
- **Reducing redundant load requests (Traffic Reduction)**
  load requests same as an inflight loading cache line are held
- **Store request compression (Traffic Reduction)**
  store requests (1e each) to a same cache line are merged
- **128B/64B variable access granularity (power reduction)**

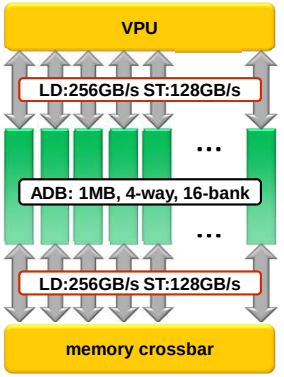
Out-of-Order Vector Memory Access

- One vector memory instruction targets 256 elements
- Consecutive/stride vector memory access: OoO by Hardware
- Indirect vector memory access: OoO by Software



Assignable Data Buffer (ADB)

- On-chip Cache for Vector
  - Private, 1MB, 4-way, 16-bank
  - 256GB/s bandwidth per core
  - Software controllable cache
  - Customized for fast random access
- Assignable Feature
  - A bypass flag in each instruction
  - Compiler/User can control
  - Avoiding cache pollution
- MSHR Feature
  - Redundant memory requests same as an inflight memory request are held to reduce memory transactions



ADB

- Compiler options
- on_adb[=category[:category...]]
- default: on_adb=arg:common:nocontiguous:
  indirect:module:reuse:stride:work
- Ftrace shows information about ADB usage !

Empowered by Innovation NEC

Performance analysis

Well known tools…

- Always use
  MPIPROGINF
  or
  F_PROGINF=

```
     ******  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
```


Well known tools…

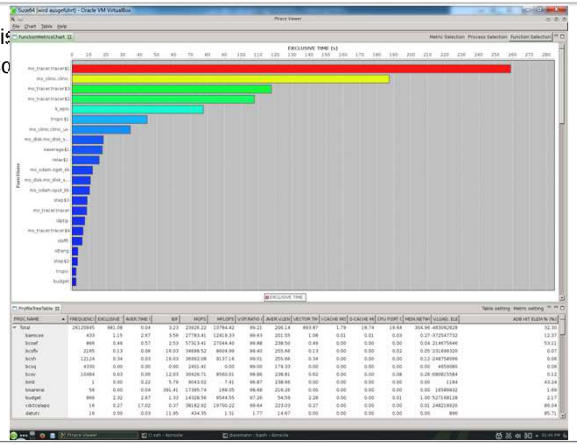
- Always use
  MPIPROGINF=DETAIL
  or
  [FC]_PROGINF=DETAIL

- Detailed profiling :
  compile and link with **-ftrace**

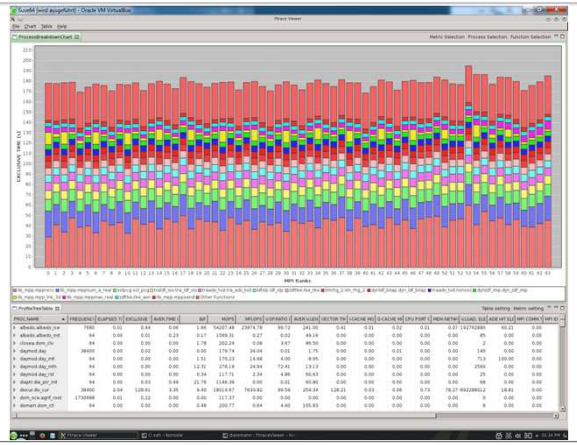
- "**ftrace**" command not available on SX-ACE nodes
  - → Save **ftrace.out*** files and evaluate with **sxftrace** on kabuki !
  - → Use env. variable **[FC]_FTRACE**

Ftrace viewer

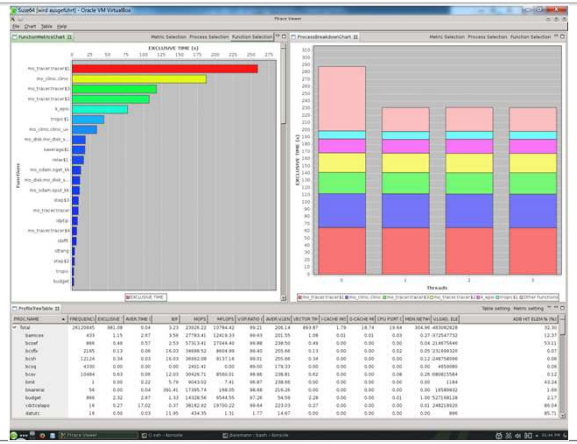
- Vis
- Co



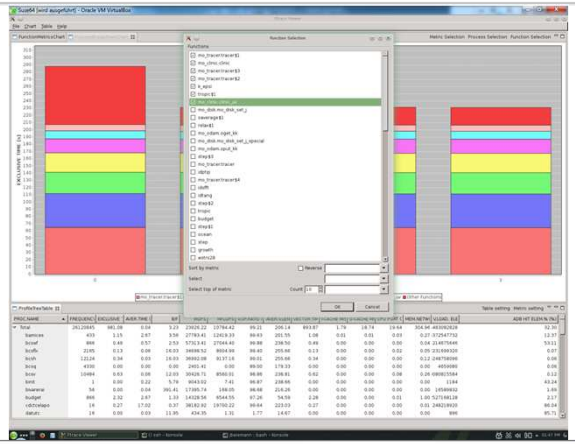
Ftrace viewer



Ftrace viewer



Ftrace viewer



Empowered by Innovation NEC

Performance, optimizations, …

Optimization for SX-ACE

- Remember your last SX / vectorization / optimization training
  - …everything still holds !
- Vectorization : not only for SX…!
- Memory access

Efficient memory access

```
      DO jk = 1, jpkm1
         DO jj = 2, jpjm1
            DO ji = fs_2, fs_jpim1
               !
               zua = - ( zwuw(ji,jj,jk) + zwuw(ji,jj,jk+1) ) / ( e1u(ji,jj) * e2u(ji,jj) &
                     * fse3u(ji,jj,jk) )
               zva = - ( zwvw(ji,jj,jk) + zwvw(ji,jj,jk+1) ) / ( e1v(ji,jj) * e2v(ji,jj) &
                     * fse3v(ji,jj,jk) )
               !
               ua(ji,jj,jk) = ua(ji,jj,jk) + zua
               va(ji,jj,jk) = va(ji,jj,jk) + zva
            END DO
         END DO
      END DO
```


Efficient memory access

```
      DO jj = 2, jpjm1
         DO jk = 1, jpkm1
            DO ji = fs_2, fs_jpim1
               !
               zua = - ( zwuw(ji,jj,jk) + zwuw(ji,jj,jk+1) ) / ( e1u(ji,jj) * e2u(ji,jj) &
                     * fse3u(ji,jj,jk) )
               zva = - ( zwvw(ji,jj,jk) + zwvw(ji,jj,jk+1) ) / ( e1v(ji,jj) * e2v(ji,jj) &
                     * fse3v(ji,jj,jk) )
               !
               ua(ji,jj,jk) = ua(ji,jj,jk) + zua
               va(ji,jj,jk) = va(ji,jj,jk) + zva
            END DO
         END DO
      END DO
```

- Identify opportunities for reusage of data in ADB

Efficient memory access

```
      INTEGER,PARAMETER:: myvl=256
      ...

      DO ii = 1,jpi,myvl

         DO jj = 2, jpjm1
            DO jk = 1, jpkm1
               DO ji = max(ii,fs_2),min(ii+myvl-1,fs_jpim1) !NEC fs_2, fs_jpim1
                  !
                  zua = - ( zwuw(ji,jj,jk) + zwuw(ji,jj,jk+1) ) / ( e1u(ji,jj) * e2u(ji,jj) &
                        * fse3u(ji,jj,jk) )
                  zva = - ( zwvw(ji,jj,jk) + zwvw(ji,jj,jk+1) ) / ( e1v(ji,jj) * e2v(ji,jj) &
                        * fse3v(ji,jj,jk) )
                  !
                  ua(ji,jj,jk) = ua(ji,jj,jk) + zua
                  va(ji,jj,jk) = va(ji,jj,jk) + zva
               END DO
            END DO
         END DO

      END DO !ii
```

- Make code more “ADB friendly”
- NEMO “dyn_zad”, “zdf_evd” : **-30%** exclusive time

Empowered by Innovation

NEC

