Empowered by Innovation

NEC
What you should learn from this course

- How to evaluate the efficiency of your application
- If efficiency is low → How to tune your code
- How can the compiler help you?
- Solve Potential Porting Problems
Terminology on Performance

**PEAK PERFORMANCE**
Theoretical value which can be reached by a computer assuming all CPU functional units are active at the same time. This is hypothetical, and will never be achieved.

**SUSTAINED PERFORMANCE**
This is the computational speed that really shows up in the system for a given code and a given problem size. It clearly relates to the architecture, because they have a different \( \rightarrow \) efficiency.

**EFFICIENCY**
Value in percent of theoretical peak performance which a system is able to deliver over any time.
Common values on PCC and MPP systems tend to be between 2 and 10%.
On Vector computers the efficiency is typically in the range of 30 to 50%.
Typical Weather Code

- SX-6(8GF/CPU), 30% efficiency
- Aladin: 30.8%

Graph showing the relationship between Peak Gflops and Effective Gflops.

- Scalar (<10% eff)

Data points:
- 32
- 48
- 64
- 320
- 240
- 160
- 100
What you should learn from this course

- How to evaluate the efficiency of your application
It's all about this table

<table>
<thead>
<tr>
<th><strong>Program Information</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Real Time (sec) : 142.373845</td>
</tr>
<tr>
<td>User Time (sec) : 122.693250</td>
</tr>
<tr>
<td>Sys Time (sec) : 1.331073</td>
</tr>
<tr>
<td>Vector Time (sec) : 65.895359</td>
</tr>
<tr>
<td>Inst. Count : 19621582641</td>
</tr>
<tr>
<td>V. Inst. Count : 2791820381</td>
</tr>
<tr>
<td>V. Element Count : 216956074637</td>
</tr>
<tr>
<td>FLOP Count : 97760445434</td>
</tr>
<tr>
<td>MOPS : 1905.449870</td>
</tr>
<tr>
<td>MFLOPS : 796.787481</td>
</tr>
<tr>
<td>VLEN : 77.711330</td>
</tr>
<tr>
<td>V. Op. Ratio (%) : 92.801205</td>
</tr>
<tr>
<td>Memory Size (MB) : 496.031250</td>
</tr>
<tr>
<td>MIPS : 159.923897</td>
</tr>
<tr>
<td>I-Cache (sec) : 1.852328</td>
</tr>
<tr>
<td>O-Cache (sec) : 15.546471</td>
</tr>
<tr>
<td>Bank (sec) : 3.069632</td>
</tr>
</tbody>
</table>

Start Time (date) : 2002/06/24 11:32:19
End Time (date) : 2002/06/24 11:34:42

- How to create
- Understand
- Improve

Use it always, It's like checking the gasoline consumption of your car ...
What you should learn from this course

- If your ported code crashes → How to solve
- If efficiency is low → How to tune your code
First principle on how to get performance

- Writing fast code is writing parallel code
- Writing parallel code on SX does not start with MPI or OpenMP!
- Single thread performance has to be improved first
- Your goal is not scalability, but time to solution
- Learn how to exploit lower levels of parallelism
- Make your code visible - the compiler will do the (most) of the rest for you, by the right directives/options
THEORY I

• Differences SX-6/SX-8
• Basics on Vectorization
NEC’s SX-Series: Innovation since 1983

SX 1/2 Series
- The first computer in the world Surpassing 1gflops

SX-3 Series
- Shared memory
- Multi-function processor
- Unix OS

SX-4 Series
- First full CMOS design
- Entirely air-cooling
- Introduction of clustered SMP architecture

SX-5 Series
- High sustained performance
- Largest ever capacity SHARED MEMORY

SX-6 Series and SX-7
- Single-chip vector processor
- TFlops scalability through optimized clustering

1983
SX 1/2 Series

1989
SX-3 Series

1994
SX-4 Series

1998
SX-5 Series

2001
SX-6 Series and SX-7

2004/5
SX-8

International Recognition
- Eckart-Mauchly Award of IEEE for CPU architecture
- IEEE and ACM Award for innovation in packaging technology
NEC’s Goals: The Vector Architecture

- Highest single CPU performance
- High sustained performance
- Extremely high bandwidth to memory
- High performance with comparatively small number of CPUs
- Same manufacturing technology like conventional CPUs
- Provide Software to use it effectively
SX-8 The Specifications

• **Single Node**
  – Up to 128 GFLOPS
    With 8 x 16 GFLOPS Processors
  – Up to 128 GBytes Shared Main Memory

• **Multi Node**
  – Up to 65 TFLOPS
  – Up to 512 Nodes Using SX-8 IXS
  – Up to 4096 Processors
  – Up to 64 TBytes Main Memory
SX-8 HLRS Installation

- 72 Nodes
- 9 TB Main Memory
- 40 TB Disk Space
- IXS with 16 GB/s bidirectional per Node
- 72 Nodes x 8 Processors x 16 GFLOPS = 9 TFLOPS
SX-8 Multi Node Configuration

16 GB/s bidirectional Bandwidth per Node
SX-8 Vector Architecture

- Multiple Vector Parallel Pipelines
- 4 Pipelines per Operation = Functional Unit
- Each Instruction Uses 4 Pipelines
- Automatic Hardware Parallelism
- Concurrent Pipeline Set Operation
- Equally Fast Division Unit
- New Square Root Unit
- More Efficient Strided Memory Access
- No Degradation for Stride Two
SX-8 Vector Unit

One Functional Unit: 2 GHz x 4 = 8 GFLOPs

- Memory
  - Vector- Data 64 x 256
  - Vregs 8x256
  - Mask
    - Vadd / Vshift
    - Vmult
    - Vlogical
    - Vdiv / Vsqrt

- Mask: 4 way pipes @ 2 GHz
- Memory: 64 GB/s


Multiplication vs. Division

\[ y(i) = x_1(i) \times x_2(i) \]

\[ y(i) = x_1(i) / x_2(i) \]

**Graphs:**
- **Left Graph:**
  - Equation: \[ y(i) = x_1(i) \times x_2(i) \]
  - Axes: MFLOPS (y-axis), Looplength (x-axis)
  - Colors: Pink (SX-8), Blue (SX-6+)

- **Right Graph:**
  - Equation: \[ y(i) = x_1(i) / x_2(i) \]
  - Axes: MFLOPS (y-axis), Looplength (x-axis)
  - Colors: Pink (SX-8), Blue (SX-6+)
Multiplication: even vs. odd stride

\[ y(i) = x_1(i) \times x_2(i) \quad i = 1, 2n, 2 \]

\[ y(i) = x_1(i) \times x_2(i) \quad i = 1, 3n, 3 \]
Square Root

$y(i) = \text{SQRT}(x(i))$

Graph showing the relationship between loop length and SQRT time for different systems:
- SX-8 (hard)
- SX-8 (soft)
- SX-6+
Scalar CPU: SX-8

- is more important than one would assume!
- features
  - 2 x 64 kByte Cache (Instructions, Data)
  - 8 kByte Instruction Buffer
  - 128 64-bit registers
  - 4 instructions per cycle
  - instruction reordering (data flow control)
  - branch prediction
  - 2 fmult/fadd/fdiv pipes, 2 integer pipes
  - double word load
  - 1 GHz -> 2 GFLOPs Peak
To transfer data at high speed between main memory and vector registers, main memory is divided into a maximum of 4096 independent modules (called banks);

Parallel reading or writing is enabled for each different bank (called interleaving or interlacing). Since only one read or write process is performed at a time in each bank, if two or more read or write requests are made to one bank, the later requests must wait for completion of the preceding request. The fall-off in speed is due to bank conflict.
Parallelization: SX-8

A SX-8 node can be used as a SMP-System or distributed memory system.

Several SX-8 nodes can be used to run a MPI-job, using 1-8 MPI-processes per node.

Current OS Version is SUPER-UX 15.1
Basics on Vectorization
SX-8 works only with IEEE Format (float0) !!!

- 4 Byte:
  - about 7 digits
  - $10^{-38} - 10^{38}$

- 8 Byte:
  - about 16 digits
  - $10^{-308} - 10^{308}$

- 16 Byte:
  - about 32 digits
  - $10^{-308} - 10^{308}$
  - not vectorizable!
Please have always in mind

<table>
<thead>
<tr>
<th></th>
<th>Real</th>
<th>Real*4</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cray</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>SX-6/8</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>SX-6/8 (-ew)</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>SX-6/8 (dbl4)</td>
<td>8</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>Workstation</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>WS (-r8)</td>
<td>8</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>SX-6/8 (idbl4)</td>
<td>8</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
Compatibility

SX-6  ok  ! -sx6

SX-7  ok  ! -sx7

SX-8
Basics on vectorization

First very important principle: segmentation of operations and pipelining
Segmentation & Pipelining

- Operations are decomposed into segments
- Example: add 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 |
Pipelining (cont.)

- no pipelining:

- with pipelining:
Segmentation Pipelining

a + b = c

Adopt Exponent

Add Mantissa

Overflow

Exponent

Normalize

a7 a8 a9 a10 a11 a12

b7 b8 b9 b10 b11 b12

a5 b5 a4 b4 a3 b3 a2 b2

c1
Pipelining (cont.)

- Superscalar pipeline (RISC):

- Vector pipeline:
Pipelining (cont.)

This is the basics of vectorization !!

4-fold parallel @ 2GHz on SX-8

Optimizations means: Keep these pipelines busy
Simple Example: Data Parallelism

- 'Vector Loop': Data Parallel
- Independent data

F77:

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

F90:

```
Real, dimension(n): a, b, c
a = b + c
```

N=256, Loop will be divided in 4 parts
SX-8 Vector Unit

Memory

Vector- Data 64 x 256

Vregs 8x256

Mask

Vadd / Vshift

Vmull

Vlogical

Vdiv / Vsqr

4 way pipes @ 2 GHz

64 GB/s
Pipelining (cont.)

Let's start with a quick test
Pipelining (cont.)

• **Vector loop: Data Parallel**

  ```plaintext
  Real, dimension(n): a, b, c
  do i = 1, n
      a(i) = b(i) + c(i)
  end do
  ```

  Each loop iteration can be executed in parallel.

• **Scalar loop: Recursion**

  ```plaintext
  Real, dimension(n): a, b, c
  do i = 1, n
      a(i) = b(i) + a(i-1)
  end do
  ```

  Current iteration depends on the result of the previous one.

  Not vectorizable.
What is peak performance of a full SX-8 node?

8*16= 128 GFLOPs

How can we calculate this?

8 (CPU) * 2 (FU) * 4 (PIP)*2 GHz

What is the maximum vector length and why is it important to use it?

256 words

Keep 4 Pipelines with 64 Elements busy
REAL, DIMENSION (51200) :: A, B, X
A = 5.25

B = 0.0

X = 10.4 + CONSTANT

B = A + X ** 2

All loops were vectorized.

note: Fortran 90 notation.
DO K = 2, KOUNT - 1
X(K+1) = X(K) + Y(K-1)
END DO

Is it vectorizable or not? Why?

Recurrence of X
Not vectorized
Vectorization

program quiz1
REAL, DIMENSION(100) :: A, B, C

DO I = 99, 1, -1
    B(I) = A(I + 1)
    A(I) = C(I)
END DO
end program

Is it vectorizable or not? Why?
Vectorization

Yes, reordering the loop index

```fortran
1  program quiz1
2    REAL, DIMENSION(100) :: A, B, C
3
4    DO I = 99, 1, -1
5
6       B(I) = A(I + 1)
7
8       A(I) = C( I )
9
10   END DO
.   !CDIR NODEP
.       do i = 1, 99
.       a(100-i) = c(100-i)
.       b(100-i) = a(101-i)
.       end do
11
12  end program
```
program quiz2
real, dimension (1000) :: a,b
    DO I = 1, 240

50 CONTINUE

    A(I) = B(I) + 2.45
    TEST = TEST + INC
    IF( TEST == SOMETHING ) GO TO 50

    END DO
end program

Does it vectorize or not? Why?
Vectorization

program quiz2
real, dimension (1000) :: a,b
    DO I = 1, 240

50 CONTINUE

    A(I) = B(I) + 2.45
    TEST = TEST + INC
    IF( TEST == SOMETHING ) GO TO 50

    END DO
end program

No - Contains a backwards branch
program quiz3
REAL, DIMENSION(1000) :: A, B

DO I = 2, 999
    B(I) = A(I - 1)
    A(I) = B(I-1)
END DO
end program

Does it vectorize or not? Why?
No, recurrence of B
End of THEORY I