Evaluating and exploiting the potential of the 2nd-generation SX-Aurora TSUBASA
Outline

Performance Evaluation
• 2nd generation SX-Aurora TSUBASA (SX-AT)

Ongoing Research Projects for SX-AT
• neoSYCL
  • A SYCL implementation for SX-Aurora TSUBASA
• Xevolver-C
  • A domain-specific language to describe vectorization-aware code tuning in C
Dancing Queen? No! It's AOBA!

What is AOBA?

- The place name of our campus (Mt. AOBA campus)
- Young growing leaves in Japanese language
System Configuration

Subsystem AOBA-A
- NEC SX-Aurora TSUBASA B401-8 x 72
  - Node Performance:
    - Comp.: 20.7 TF
    - Mem size: 640 GB
    - Mem BW: 12.4 TB/s
- AMD EPYC 7402P x 1
- NEC Vector Engine Type 20B x 8
- Vector Engine Core x 8
- HBM2 Module x 6

Subsystem AOBA-B
- NEC LX 406Rz-2 x 68
  - Node Performance:
    - Comp.: 4.1 TF
    - Mem size: 256 GB
    - Mem BW: 0.41 TB/s
- AMD EPYC 7702P x 2

Overall System Performance: 1.78 Pflop/s, 924 TB/s
## Evaluation Setup

<table>
<thead>
<tr>
<th></th>
<th>SX-ACE</th>
<th>SX-AT 10B</th>
<th>SX-AT 10AE</th>
<th>SX-AT 20B</th>
<th>EPYC 7702</th>
<th>Xeon 6246</th>
<th>Xeon 6248</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Core</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clock Freq.</td>
<td>1</td>
<td>1.4</td>
<td>1.58</td>
<td>1.6</td>
<td>2</td>
<td>3.3</td>
<td>2.5</td>
</tr>
<tr>
<td>[GHz]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Peak Perf.</td>
<td>64</td>
<td>268</td>
<td>304</td>
<td>307</td>
<td>32</td>
<td>105</td>
<td>80</td>
</tr>
<tr>
<td>[Gflop/s]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Socket</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of Cores</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>64</td>
<td>12</td>
<td>20</td>
</tr>
<tr>
<td>Peak Perf.</td>
<td>256</td>
<td>2150</td>
<td>2433</td>
<td>2457</td>
<td>2048</td>
<td>1267</td>
<td>1600</td>
</tr>
<tr>
<td>[Gflop/s]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mem BW</td>
<td>256</td>
<td>1228</td>
<td>1352</td>
<td>1536</td>
<td>204</td>
<td>140</td>
<td>140</td>
</tr>
<tr>
<td>[GB/s]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Node</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of Sockets</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Peak Perf.</td>
<td>256</td>
<td>2150</td>
<td>2433</td>
<td>2457</td>
<td>4096</td>
<td>2534</td>
<td>3200</td>
</tr>
<tr>
<td>[Gflop/s]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System B/F</td>
<td>1</td>
<td>0.56</td>
<td>0.56</td>
<td><strong>0.62</strong></td>
<td>0.1</td>
<td>0.11</td>
<td>0.09</td>
</tr>
</tbody>
</table>
HPL Performance

Sustained Perf. / Peak Perf.

1 socket (1 VE)

2 sockets (2 CPUs)
HPL Power Efficiency

![Graph showing HPL Power Efficiency for different processors](image-url)
HPCG Performance

![Graph showing HPCG Performance and Peak Memory Bandwidth]

- HPCG [Gflop/s]
- Peak BW [GB/s]

<table>
<thead>
<tr>
<th>System</th>
<th>Performance [Gflop/s]</th>
<th>Peak Memory Bandwidth [GB/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>SX-AT 20B</td>
<td>140</td>
<td>1800</td>
</tr>
<tr>
<td>EYPC 7702</td>
<td>80</td>
<td>1600</td>
</tr>
<tr>
<td>Xeon 6246</td>
<td>40</td>
<td>1200</td>
</tr>
<tr>
<td>Xeon 6248</td>
<td>20</td>
<td>800</td>
</tr>
</tbody>
</table>

31st Workshop on Sustained Simulation Performance (online)
HPCG Power Efficiency

![Graph showing HPCG Power Efficiency](image)

**Sustained Perf. / Peak Perf.**
HPCG & HPL on AOBA

Subsystem AOBA-A

**NEC SX-Aurora TSUBASA B401-8 x 72**

- **Node Performance**
  - Comp. : 20.7 TF
  - Mem size : 640 GB
  - Mem BW : 12.4 TB/s

**Subsystem AOBA-B**

**NEC LX 406Rz-2 x 68**

- **Node Performance**
  - Comp. : 4.1 TF
  - Mem size : 256 GB
  - Mem BW : 0.41 TB/s

**【SX-Aurora】**
72VH(576VE) Peak 1415.6TF

- HPL : 1140.1TF (Efficiency 80.5%)
- HPCG : 75.3TF (Efficiency 5.32%)

**【AMD Rome 7702】**
68node Peak 278.5TF

- HPL : 161.7TF (Efficiency 58.08%)
- HPCG : 2.55TF (Efficiency 0.92%)

Egawa et al.@PMBS20
Key Applications

- **FDTD-MAS (Konno et al. 2016)**
  - Analysis of noises on a PCB board

- **Reflext_Array (Kasuga et al. 2005)**
  - Simulation of a reflect array

- **Heat_Stroke (Kojima et al. 2018)**
  - Evaluation of the heat stroke risk

- **Nano_Powder (Shigera et al. 2018)**
  - Simulation of generating silicon nanoparticles

- **MHD (Fukazawa et al. 2016)**
  - Simulation of radiation band electrons around Jupiter

- **Turbine (Yamamoto et al. 2011)**
  - Design of steam and gas turbines

- **Multiphase_Fl (Yoshida 2020)**
  - Simulation of dust explosion phenomena
## Application Summary

<table>
<thead>
<tr>
<th></th>
<th>Vectorization [%]</th>
<th>Ave Vec Len</th>
<th>No. of Processes</th>
<th>No. of VEs</th>
<th>B/F</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPL</td>
<td>99.3</td>
<td>254.4</td>
<td>256</td>
<td>32</td>
<td>0.31</td>
</tr>
<tr>
<td>HPCG</td>
<td>99.0</td>
<td>237.5</td>
<td></td>
<td>32</td>
<td>11.08</td>
</tr>
<tr>
<td>FDTD-MAS</td>
<td>99.6</td>
<td>252.7</td>
<td>1</td>
<td>1</td>
<td>0.29</td>
</tr>
<tr>
<td>Reflect_Array</td>
<td>98.8</td>
<td>216.1</td>
<td>8</td>
<td>1</td>
<td>3.74</td>
</tr>
<tr>
<td>Heat_Stroke</td>
<td>99.3</td>
<td>214.2</td>
<td>160</td>
<td>20</td>
<td>1.87</td>
</tr>
<tr>
<td>Nano_Powder</td>
<td>98.9</td>
<td>255.1</td>
<td>32</td>
<td>32</td>
<td>2.40</td>
</tr>
<tr>
<td>MHD</td>
<td>99.6</td>
<td>249.6</td>
<td></td>
<td>32</td>
<td>1.33</td>
</tr>
<tr>
<td>Turbine</td>
<td>92.0</td>
<td>175.5</td>
<td>159</td>
<td>20</td>
<td>2.52</td>
</tr>
<tr>
<td>Multiphase_Fl</td>
<td>29.1</td>
<td>18.0</td>
<td>1</td>
<td>1</td>
<td>12.88</td>
</tr>
</tbody>
</table>

*Compute-intensive*  
*Memory-intensive*
Performance Evaluation Results

SX-ACE: $V\% = 99.407\%, V_L = 237.95$
SX-AT: $V\% = 99.632\%, V_L = 252.65$

SX-ACE: $\$hit = 15.6\%$
SX-AT: $\$hit = 34.2\%$

Compute-intensive
Memory-intensive
Summary so far

- Supercomputer AOBA

  - Start operation on **October 1, 2020**.
  - **The world’s first system** of 2nd Generation SX-Aurora TSUBASA
    - Standard Software + Special Hardware
    - High Memory Bandwidth
    - Vector Engine Type 20B
      → Higher B/F Rate = **Higher Comp.&Power Efficiencies**
neoSYCL

A SYCL implementation for SX-Aurora TSUBASA
Motivation

- VH and VE are general-purpose processors
  - Stand-alone, general-purpose processors
  - Fully supporting standard languages, e.g. C/C++/Fortran
  - Different performance characteristics
    - VE works much faster only for vector-friendly workloads

The right processor should be assigned to the right task.
Related Work

**Vector Engine Offloading (VEO) and VH Call**

- VH-VE collaboration
  - Data transfers
  - Function calls
  - Synchronization
  - VH-initiated (VEO) and VE-initiated (VH call)

**X** NEC’s proprietary programming tools
= application code needs to be specialized only to NEC SX-Aurora TSUBASA

Open standard for offload programming = SYCL
SYCL Single Source C++ Parallel Programming

C++ Kernel Fusion can give better performance on complex apps and libs than hand-coding.

Accelerated code passed into device OpenCL compilers.

SYCL is ideal for accelerating larger C++-based engines and applications with performance portability.

Complex ML frameworks can be directly compiled and accelerated.

C++ templates and lambda functions separate host & accelerated device code.
SYCL Implementations in Development

SYCL, OpenCL and SPIR-V, as open industry standards, enable flexible integration and deployment of multiple acceleration technologies.

SYCL enables Khronos to influence ISO C++ to (eventually) support heterogeneous compute.

Multiple Backends in Development

SYCL beginning to be supported on multiple low-level APIs in addition to OpenCL e.g., ROCm and CUDA

For more information: [http://sycl.tech](http://sycl.tech)
Kernel Outlining

Simple Triad implementation

```c
for (int i = 0; i < array_size; i++) {
    b[i] = scalar * c[i];
}
```

SYCL Triad implementation

```c
queue.submit([&](handler &cgh) {
    auto ka = buf_a.template get_access<access::mode::write>(cgh);
    auto kb = buf_b.template get_access<access::mode::read>(cgh);
    auto kc = buf_c.template get_access<access::mode::read>(cgh);
    cgh.parallel_for<class triad_kernel>(range<1>(array_size), [=](const id<1> &i) {
        ka[i] = kb[i] + scalar * kc[i];
    });
});
```

Kernel Outlining by neoSYCL

```c
int triad_kernel(double *ka, double *kb, double *kc, int N) {
    for (int i = 0; i < N; i++) {
        ka[i] = kb[i] + 0.4 * kc[i];
    }
    return 0;
}
```

A kernel is translated into a loop in a separate function

→ VEO can invoke it on VE.
Performance Evaluation

- BFS (Rodinia)

*Programming with SYCL leads to lower code complexity*
Xevolver-C
A DSL for describing performance engineering
Xevolver Framework

Various transformations are required for replacing **arbitrary code modifications**. = cannot be expressed by combining predefined transformations.

→ Xevolver: a framework for custom code transformations

**Predefined or user-defined annotations**

**App code**

**s2s translator**

**Optimized for System A**

**Optimized for System B**

**Translation rules**

- Define the code transformation of each annotation
- Different systems can use different rules
- Users can define their own code transformations
Xevolver-C

- DSL for user-defined code transformation
  - C language syntax with some special features

```c
int i,j;
xev_stmt* any_stmt;

int main(int argc, char** argv){
  #pragma xev stmt src("label1")
  {
    for(i=0;i<10;i++){
      for(j=0;j<10;j++){
        any_stmt;
      }
    }
  }
  #pragma xev stmt dst("label1");
  {
    for(int ij=0;ij<100;ij++){
      i=ij%10;
      j=ij/10;
      any_stmt;
    }
  }
}
```

Original loop structure

```
for(i=0;i<10;i++){
  for(j=0;j<10;j++){
    any_stmt;
  }
}
```

All the statements in the loop body are copied

Transformed loop structure

```
for(int ij=0;ij<100;ij++){
  i=ij%10;
  j=ij/10;
  any_stmt;
}
```

Code patterns are written in the rule \(\rightarrow\) **easily customizable** for individual cases
Performance Evaluation

**HPC refactoring catalog (Egawa et al. 2017)**

- A collection of 33 code optimization cases
  - Effective for SX-9 and (probably) SX-ACE
  - Sample codes are written in Fortran

- 23 optimizations have been translated into C
- 22 optimizations can be expressed by Xev-C
- 20 optimizations can improve SX-AT performance

**How about SX-AT?**
How about C?

Performance did not change in 2 cases because the newer compiler for SX-AT works well even for the original version.
= Optimization not needed for the 2 cases.
... and many more

Other ongoing projects
“Quantum Compiler”

Loop vectorization needs to solve a combinatorial problem (NP-hard)

A part of the compilation process is offloaded to D-Wave machine

A bipartite graph is converted to QUBO

Sent to QA

A solution is sent back

Sasaki et al.@ISC20
Abstraction of conflict avoiding

Automatic data padding to avoid conflicts

We can avoid spike-like performance drops

The constructor of `abc::array` is called.

Arguments
- `array size (integer): Ni, Nj, ...`
- `reference of abc::group : ref_group`

Allocate for `array size + margin`.

`ptr_org`  
`array size + margin`

The `abc::group` assigns a bank ID.

`abc::group ref_group`

Bank ID

Calculate the padding size `p` and pad `ptr_org`.

`ptr_org`  
`padding size p`  
`ptr_org+p`

Then, `ptr_org+p` is the initial address of the array region.

Figure 8. The evaluation results of UPACS-Parts.
Urgent Computing

- Use of an HPC system at Emergency
  - Preemptive execution of urgent jobs on a heterogeneous system (such as AOBA)

- Scheduling urgent jobs while keeping the efficiency of regular jobs
Summary

Lately we have started many research projects for NEC SX-Aurora TSUBASA

- Performance evaluation (AOBA)
- Programming model (neoSYCL)
- Performance tuning tool (Xevolver-C)
- Urgent computing
- Job scheduling
- Machine learning
- Simulated annealing ...

Stay tuned, there is more to come!
Acknowledgments

Many thanks to the collaborators.

- Ryusuke Egawa (Tokyo Denki University)
- Souya Fujimoto (NEC)
- Yoko Isobe (NEC)
- Yoichi Shimomura (NEC)
- Tsuyoshi Yamashita (CC, Tohoku University)
- Daisuke Sasaki (CC, Tohoku University)
- Joint-Research Division of High Performance Computing (NEC), CC, Tohoku University
  Led by Professor Hiroaki Kobayashi

This work is partially supported by MEXT Next-Generation High-Performance Computing Infrastructures and Applications R&D Program “R&D of a Quantum-Annealing-Assisted Next Generation HPC Infrastructure and its Applications,” and Grant-in-Aid for Scientific Research(A) #20H00593.