Automatically Avoiding Memory Access Conflicts on SX-Aurora TSUBASA

Naoki Ebata*, Ryusuke Egawa†*, Yoko Isobe‡, Ryoji Takaki§, Hiroyuki Takizawa†*

* Graduate School of Information Sciences, Tohoku University, naoki.ebata.r7@dc.tohoku.ac.jp
† Cyberscience Center, Tohoku University, {egawa, isobe, takizawa}@tohoku.ac.jp
‡ NEC Corporation
§ Japan Aerospace Exploration Agency, ryo@isas.jaxa.jp
Outline

- Introduction
- Proposed metric
- Proposed C++ library
- Performance evaluation
- Conclusions and future work
Outline

- Introduction
- Proposed metric
- Proposed C++ library
- Performance evaluation
- Conclusions and future work
Background

- Recently, the performance of many scientific applications is limited by the sustained memory bandwidth. → Strong demands for a computing system that can achieve high sustained memory bandwidth.

- SX-Aurora TSUBASA Vector Engines (VEs) have world’s top-class theoretical memory bandwidth.
  - SX-Aurora TSUBASA VEs: modern vector processors.
  - The theoretical memory bandwidth of a VE → 1.2 TB/s.
  - They are highly suited to execute memory-intensive applications.

https://jpn.nec.com/hpc/sxauroratsubasa/features/index.html?
Memory Access Conflicts

- Although VEs have a high theoretical memory bandwidth, memory access conflicts degrade the sustained memory bandwidth.
  - A memory access conflict means that multiple accesses to a specific memory bank or port occur at the same time and conflict.
  - They degrade the sustained memory bandwidth.

- Large negative effects of memory access conflicts are observed on VEs.
Preliminary Evaluation

 Executing vector add operations vector add (following code) on a VE (single core) while changing its vector length.

```
for(int i = 0; i < N; i++){
    a[i] += b[i];
}
```

At some array sizes, sustained memory bandwidths drop significantly due to access conflicts.
Traditional techniques

- Padding is a widely used code tuning technique to avoid memory access conflicts.

- By inserting extra elements to an array, optimizing the memory access patterns.
  - Example: `double a[N][M] \xrightarrow{\text{\textend{inline}} double a[N][M+p]`  

- However, the code optimization and tuning of the padding size $p$ needs to be manually done by programmers, and it is a tedious task for programmers.
Our Goal & Approach

Goal

- Automatically avoid memory access conflicts on modern vector processors (VEs).
  - There are some related works for automatically optimizing memory accesses to avoid access conflicts on CPU/GPU. For example, Shuang Gao et. al has reported the effective method to avoid bank conflicts on a CUDA architecture.
    - However, the memory architecture of modern vector processors is totally different from them. Thus, they cannot be applied to VEs directly.

Approach

- Predicting memory accesses conflict using a simple metric.
- Implementing an array-like C++ library "abc" that allocates a memory chunk so as to prevent memory access conflicts.
Outline

- Introduction
- **Proposed metric**
- Proposed C++ library
- Performance evaluation
- Conclusions and future work
Main memory configuration of VEs

- Main memory consists of multiple memory modules and channels and banks.
  - Modules: 6 HBM2 module / VE
  - Channels: 8 channels / module
  - Banks: 32 banks / channel (VE Type 10B)
How data are allocated to memory banks?

- A sequence of memory addresses is assigned to physical memory banks by round-robin for memory allocation.

: The $b$-th bank of the $c$-th channel in the $m$-th memory module
Proposed metric

- A simple metric $d$ to predict access conflicts between two arrays.
  - In this paper, we focus on bank conflicts between arrays.
- $d$ is defined as following equation.

$$d = \left( \left\lfloor \frac{ADR_{Sa}}{S_{cell}} \right\rfloor - \left\lfloor \frac{ADR_{Sb}}{S_{cell}} \right\rfloor \right) \mod N_{bank}.$$ 

- $ADR_{Sa}, ADR_{Sb}$: The addresses of a head element of array a, b (&a[0], &b[0])
- $N_{bank}$: Total number of memory banks.
- $S_{cell}$: Size of memory cell. Memory cell is a continuous data region handled by VEs as one unit.
What the metric means?

- Simply stated, $d$ means the distance between two arrays.
  - It is assumed that bank conflicts frequently occur in specific distances.

- Example:
  - In following figure, $d_{ac} = 1$, $d_{ad} = 5$, $d_{cd} = 4$, $d_{ae} = 48(=6\times8)$, $d_{ce} = 47$, ...
  - If head elements of two arrays are assigned to same bank ($a$, $b$ in following figure), then $d_{ab} = 0$. 

![Diagram illustrating the concept of distance between arrays and bank conflicts.]

\( m = 0 \)
\( b = 0 \ 1 \dots \ 31 \)
\( c = 0 \)
\( a[0], b[0] \)
\( c[0] \)
\( d[0] \)
\( m = 1 \)
\( b = 0 \ 1 \dots \ 31 \)
\( c = 1 \)
\( e[0] \)
\( m = 5 \)
\( b = 0 \ 1 \dots \ 31 \)
Relationship between $d$ and performance

- We investigate the relationship for vector add kernel.

- It can be seen that $d$ is a good representation of the characteristics of bank conflicts.
  - There is an obvious tendency that the sustained memory bandwidth degrades when $d$ is approximately a multiple of 512.
Outline

- Introduction
- Proposed metric
- **Proposed C++ library**
- Performance evaluation
- Conclusions and future work
Proposed C++ class library “abc”

- **abc consists of two classes abc::group and abc::array.**
  - abc::array: An array can be replaced with abc::array.
  - abc::group: Multiple instances of abc::array are grouped by using an instance of abc::group.
The role of `abs::group`

- Our library allocates the memory to each `abc::array` instances so as not to occur access conflicts between `abc::array` instances belonging to a same `abc::group`.

![Diagram showing memory allocation and access conflicts between `abc::array` instances in different `abc::group`]

- `abc::array a`
- `abc::array b`
- `abc::array c`
- `abc::group g1`
- `abc::array d`
- `abc::group g2`

No conflicts

Maybe conflict
Example

Sample implementation of vector add using abc.

```c
int main(){
    constexpr int N = 100;
    abc::group sample;
    abc::array<double> a(N, sample), b(N, sample);

    // initialize
    for(int i = 0; i < N; i++){
        a(i) = 1.0;
        b(i) = 2.0;
    }

    // vector add
    for(int i = 0; i < N; i++) b(i) += a(i);
}
```
How it works

1. The constructor of `abc::array` allocates the memory region of (array size + margin) bytes.
2. `abc::group` assigns a bank.
3. Calculate the padding size and pad the pointer.

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

Arguments
- array size (integer): $N_i, N_j, \ldots$
- reference of `abc::group : ref_group`

`malloc` for array size + margin.

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

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

Then, `ptr_org+p` is the initial address of the array region.
The rule of bank assignments

For the \( i \)-th \( \texttt{abc::array} \) instance, \( \texttt{abc::group} \) assigns a bank according to the following rule.

\[
\text{ID}(i) = \begin{cases} 
0 & (i = 1) \\
N_{bank} \times \left\{ 2 \times \left\{ (i-1) \mod 2^{\log_2(i-1)} \right\} + 1 \right\} & (i \neq 1), 
\end{cases}
\]

- ID means that the number of banks to be assigned counting from the 0-th bank of 0-th channel of 0-th module.
- For example, the ID of 0-th bank of 0-th channel of 0-th module is 0, the ID of 0-th bank of 0-th channel of 1-th module is 1, the ID of 31-th bank of 7-th channel of 5-th module is 1535.

The objective of this rule is to assign banks so as not to exist any pair of \( \texttt{abc::array} \) instances whose \( d \) is close to multiple of 512.
Outline

- Introduction
- Proposed metric
- Proposed C++ library
- Performance evaluation
- Conclusions and future work
Experimental setup

- **Benchmarks**
  - Vector add
  - UPACS-Parts (CFD application kernels)

- **System specification**

<table>
<thead>
<tr>
<th></th>
<th>SX-Aurora TSUBASA A300-8</th>
</tr>
</thead>
<tbody>
<tr>
<td>VE Type</td>
<td>Type 10B</td>
</tr>
<tr>
<td>Compiler</td>
<td>nc++ (NCC) 2.5.1</td>
</tr>
<tr>
<td>VEOS</td>
<td>veos-2.2.2-1.el7.x86 64</td>
</tr>
<tr>
<td>Theoretical memory bandwidth</td>
<td>1.20 (TB/s)</td>
</tr>
<tr>
<td>Theoretical computational performance</td>
<td>2.15 (TFLOPS)</td>
</tr>
</tbody>
</table>
Vector add

- We evaluated the proposed method by vector add.
  - We run both normal and abc code with single core.
- Our method can successfully avoid performance degradations caused by memory access conflicts.
- The average performance of abc implementation is lower than one of the original code.
  - It is side effect of data rearrangement of abc.
  - Since the vector add is a quite simple kernel which uses only two arrays, our data rearrangement may cause a data congestion on a specific channel.
UPACS-Parts

- UPACS-Parts are kernel codes used in UPACS, which is a CFD code for aerospace applications that has been developed by JAXA (Japan Aerospace eXploration Agency).

- We evaluated our method for four kernels:
  - Streaming type: cflux, vflux
  - Stencil type: cfacev, muscl
  - These performance is limited by the sustained memory bandwidth.

- Since the original codes of UPACS-Parts is written in Fortran, we rewrote them in C++.

- They are parallelized with OpenMP and we run them with 8 core.
Streaming type: cflux, vflux

- At the most array sizes, our method mitigates performance degradations.
Stencil type: muscl, cfacev

- As in streaming type, our method successfully avoids performance degradations.
Remaining performance degradations

In both of streaming type and stencil type, there are some array sizes where small performance degradations exists despite the facts that our method is applied.

There are 2 possible reasons:

- Both of them are not took into account in abc.
  1. Multiple accesses from other threads conflict.
  2. Accesses to the elements of a same array conflict.
Outline

- Introduction
- Proposed metric
- Proposed C++ library
- Performance evaluation

Conclusions and future work
Conclusions & Future Work

Conclusions

- Intra-array bank conflicts can be avoided by tuning the relative distance between two arrays that is assumed to be accessed at the same time.
  - We introduced a metric representing the distance in terms of memory bank.
- This tuning method is automated as an array-like C++ class.
- We demonstrated our method can mitigate negative effects of memory access conflicts on CFD application kernels.

Future Work

- We will implement our approach in the form of widely-used libraries for VEs.
  - Now, our proposed method is specialized for VEs. Thus, it is not portable to other platforms.
Acknowledgments

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”
- Grant-in-Aid for Scientific Research(B) #16H02822 and #17H01706