\Orchestrating a brighter world NEC

VEDA: Best practices to use hybrid programming on the **NEC SX-Aurora TSUBASA**

Dr. Nicolas Weber
NEC Laboratories Europe

Agenda

1. Using C++ function overloading and templates with VEDA
2. Improving performance of memset operations
3. Improving performance of memcopy operations

Using C++ function overloading and templates with VEDA

Fetching C-style Functions

Device

```
extern “C” int func(int A) { ... }
```

Host

```
VEDAfunction func;
vedaModuleGetFunction(&func, mod,
“func”);
```

Fetching C++-style Functions

Device

```
int   func(int A)   { ... }
float func(float A) { ... }
```

Host

```
VEDAfunction func;
vedaModuleGetFunction(&func, mod,
"_Z4funci");
vedaModuleGetFunction(&func, mod,
"_Z4funcf");
```

[Small Detour] What is Name Mangling?

- C++ is “only” an extension to C and needs to be C-ABI compatible
- C-ABI only supports [A-Za-z0-9_] for names
- C++ names:
  - _[Type][Name][Template][Arguments]
  - [Type] `Z` = `Function`
  - [Name] `my_namespace::func3a` = `N12my_namespace6func3aE`
  - [Template] `template<void, float*, 5>` = `IvPfLi5EE`
  - [Arguments] `float, float*, float*` = `fPfS0_`
- Can become very complex:
  `_ZN2ns4veda13veda_templateIN3bla7complexIfEEEEvT_S5_`
- Further Reading:
  - https://github.com/gchatelet/gcc_cpp_mangling_documentation

[Solution 1] extern “C” function wrapper

Device

```
int   func(int A)   { ... }
float func(float A) { ... }

#define WRAP(T)¥
extern “C” func_##T(T A) {¥
   return func(A);¥
}
WRAP(int)
WRAP(float)
```

Host

```
VEDAfunction func;
vedaModuleGetFunction(&func, mod,
“func_int”);
vedaModuleGetFunction(&func, mod,
“func_float”);
```

[Solution 2] Dispatching function

Device

```
template<int A, int B, typename T>
void gemm(...) { ... }

extern "C" void gemm(int A, int B,
int T, ...) {
   if(A == 2) {
      if(B == 3) {
         if(T == 0)
            gemm<2, 3, float>(...);
   ...
}
```

Host

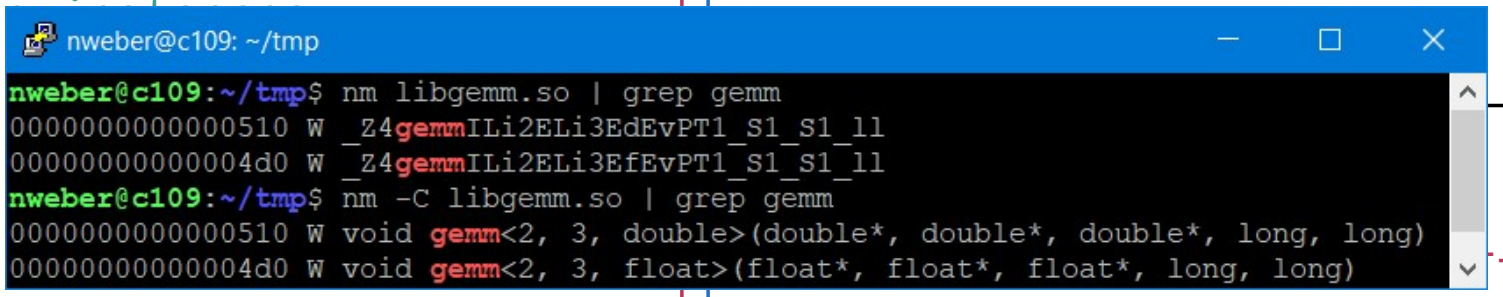
```
VEDAfunction func;
vedaModuleGetFunction(&func, mod,
"gemm");
```

[Solution 3] Mangled name lookup

Device

```
template<int A, int B, typename T>
void gemm(...) { ... }

// instantiat
template void
template void
```



Host

```
template<int A, int B, typename T>
static constexpr const char*
gemm(void) {
   if constexpr (A == 2) {
                                    _same_v<T,
                                    _ll";
    ...
}

VEDAfunction func;
vedaModuleGetFunction(&func, mod,
gemm<2,3,float>());
```

[Preview] Upcoming experimental VEDA C++ API

```
extern “C” int func(int A) { ... }

int   func(int A)   { ... }
float func(float A) { ... }

template<int A, int B, typename T>
void gemm(...) { ... }

// instantiate instances
template void gemm<2,3,float> (...);
template void gemm<2,3,double>(...);
```

```
using namespace veda;

auto A = CFunction::Return<int>(mod,
“func”);
auto B =
Function::Return<int>::Args<int>(mod,
“func”);
auto C = Template<Literal<2>,
Literal<3>, float>::Args<...>(mod,
“gemm”);
```

will be available in next Github release

\Orchestrating a brighter world NEC

More details in our SX-Aurora Article:

VEDA: Best practices to use hybrid programming on the NEC SX-Aurora TSUBASA

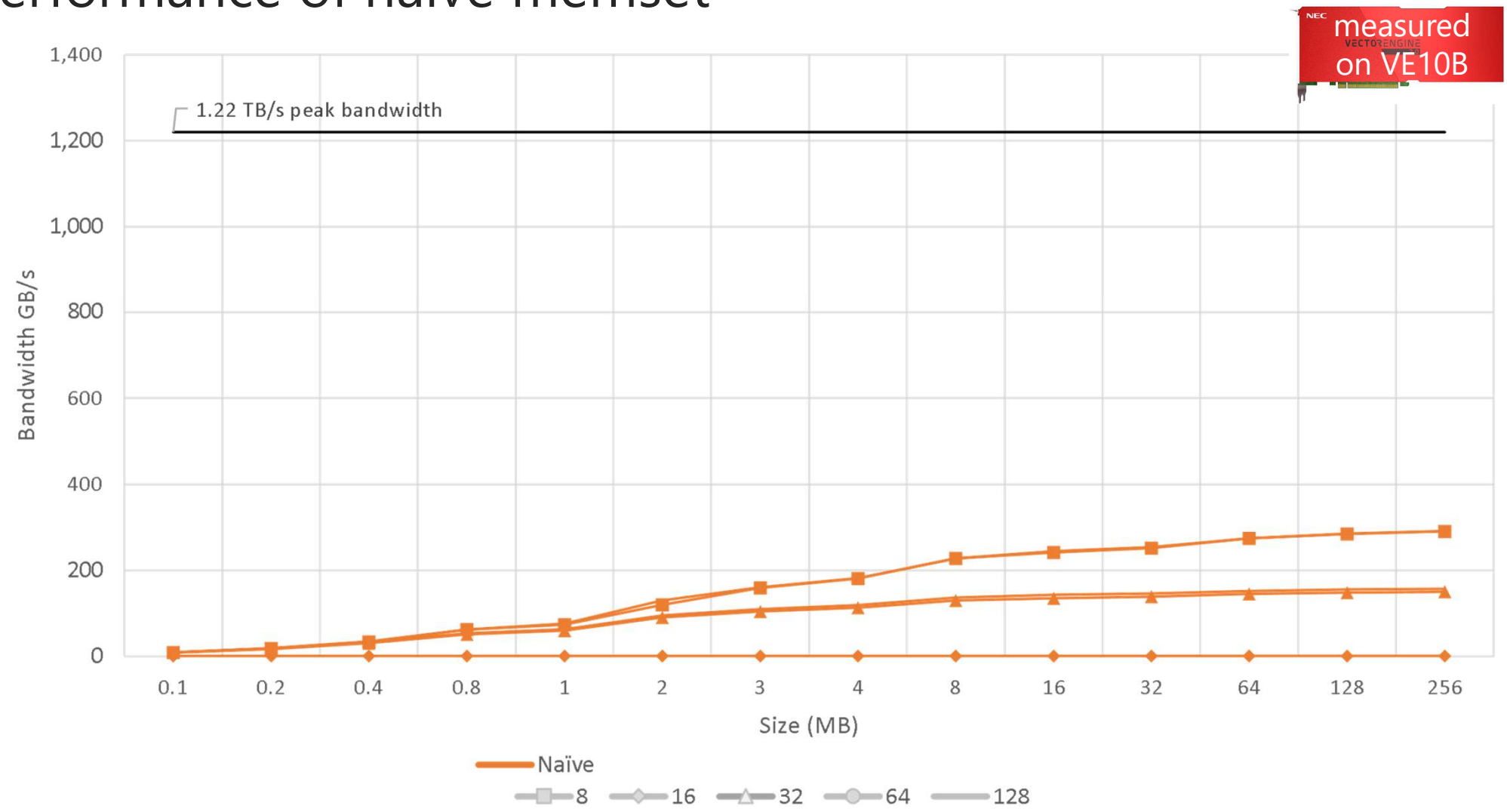
Improving performance of memset operations

Naive implementation

- Memset is a rather simple operation:

```
template<typename T>
void naïve_memset(T* dst, const T value, const size_t cnt) {
   #pragma _NEC select_vector
   for(size_t i= 0; i < cnt; i++)
      dst[i] = value;
}
```

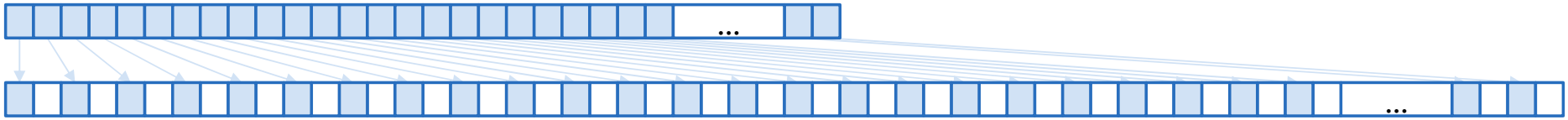
Performance of naïve memset



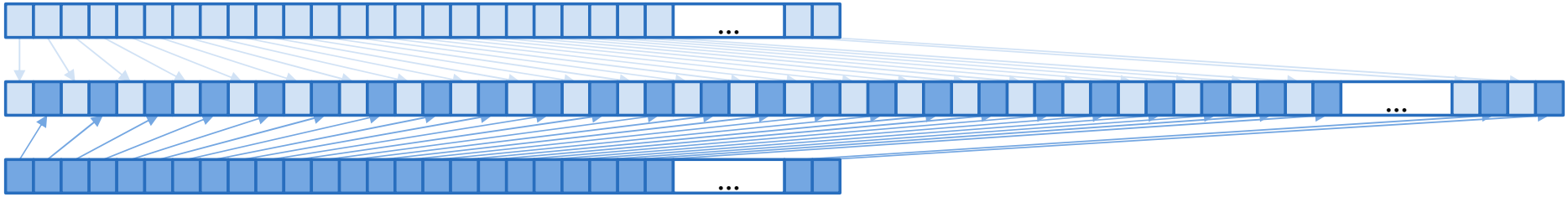
Why is the performance so differently?

- 16-bit data types are not vectorizable
- 32-bit (packed vector) suffer from interleaving
  - VE has vector registers with 256x 64-bit

  - Simple Float vectorization places one 32-bit value into one 64-bit register



  - Packed Float vectorization places two 32-bit values into one 64-bit register



Naive implementation

- Memset is a rather simple operation:

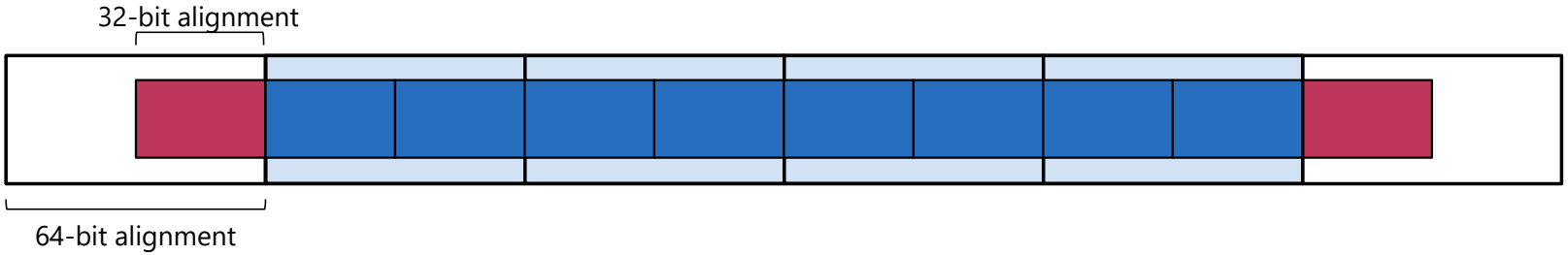
```
template<typename T>
void naïve_memset(T* dst, const T value, const size_t cnt) {
   #pragma _NEC select_vector
   for(size_t i= 0; i < cnt; i++)
      dst[i] = value;
}
```

[Solution] Use 64-bit for all data types

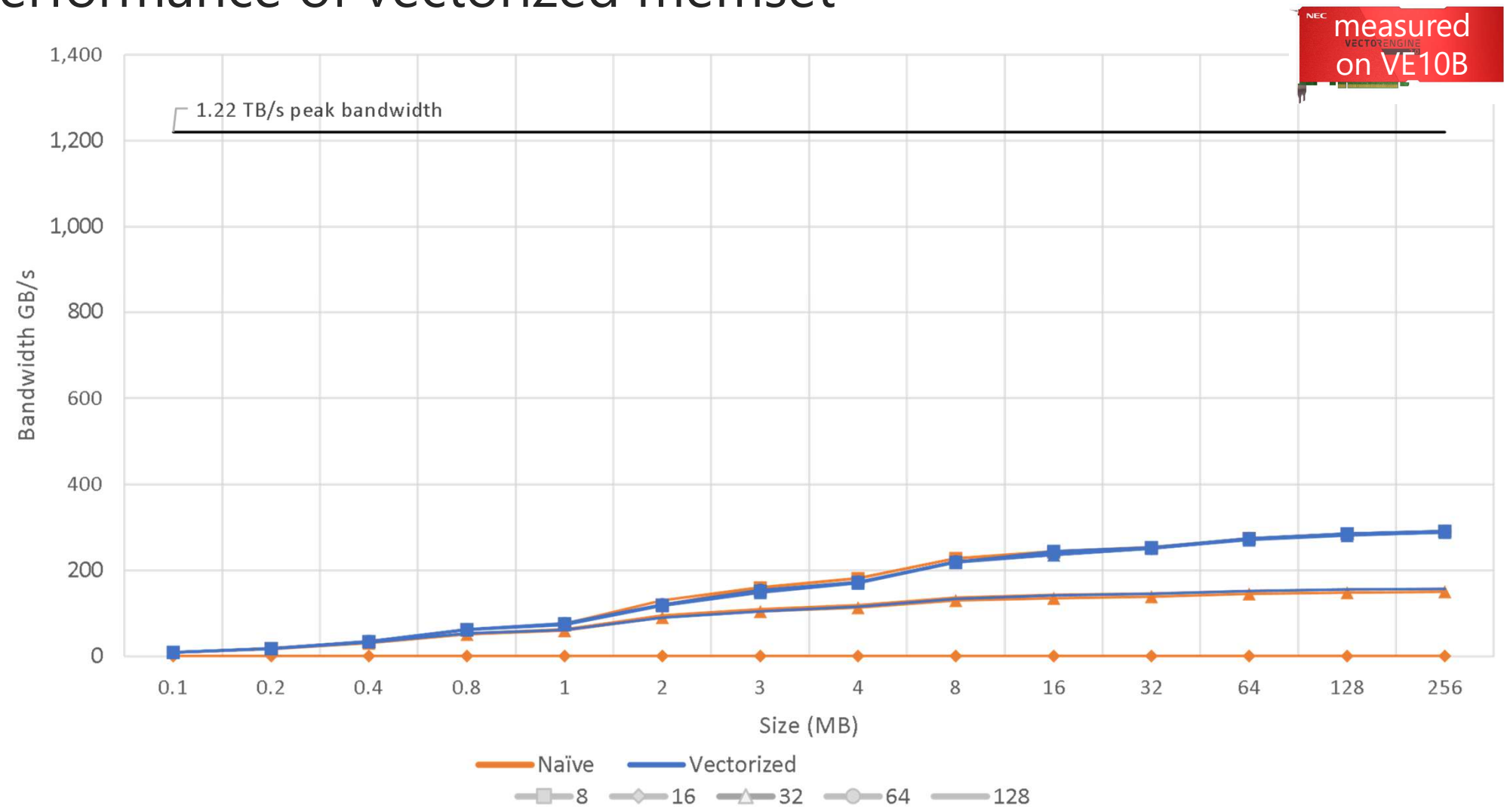
- **Problem #1:** How to convert 8, 16 or 32-bit values to 64-bit?

```
const T vx[]       = {v, v, v, v, v, v, v};
const int64_t v64 = *(const int64_t*)&vx;
```

- **Problem #2:** misaligned memory access
  - N-Bit variables, need to be N-Bit memory aligned



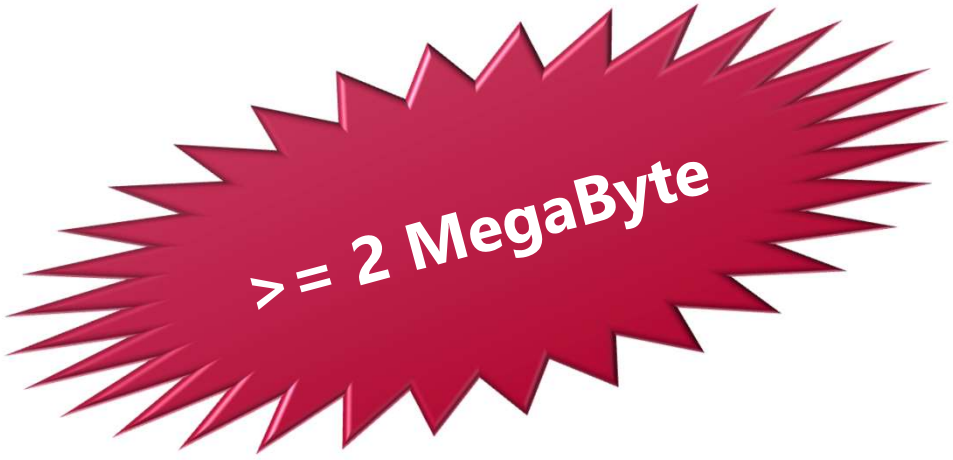
Performance of vectorized memset



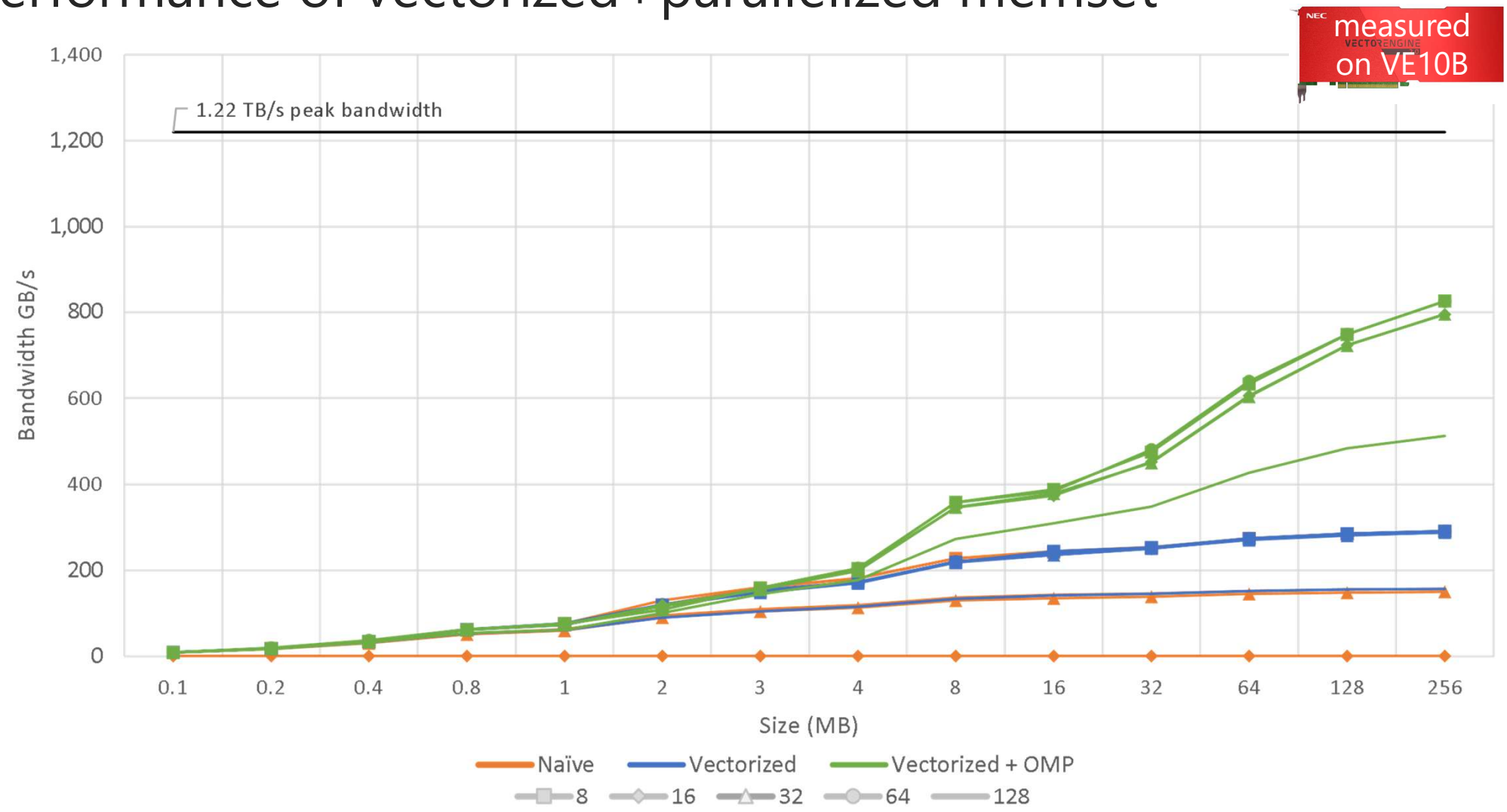
[Solution] Use multiple cores!

- Just do #pragma omp parallel for?
- Unfortunately, the OpenMP launch overhead kills performance for small memsets

- After lengthy and very thorough performance measurements we identified the extremely complex correlation for optimal memset performance using OpenMP:



Performance of vectorized+parallelized memset



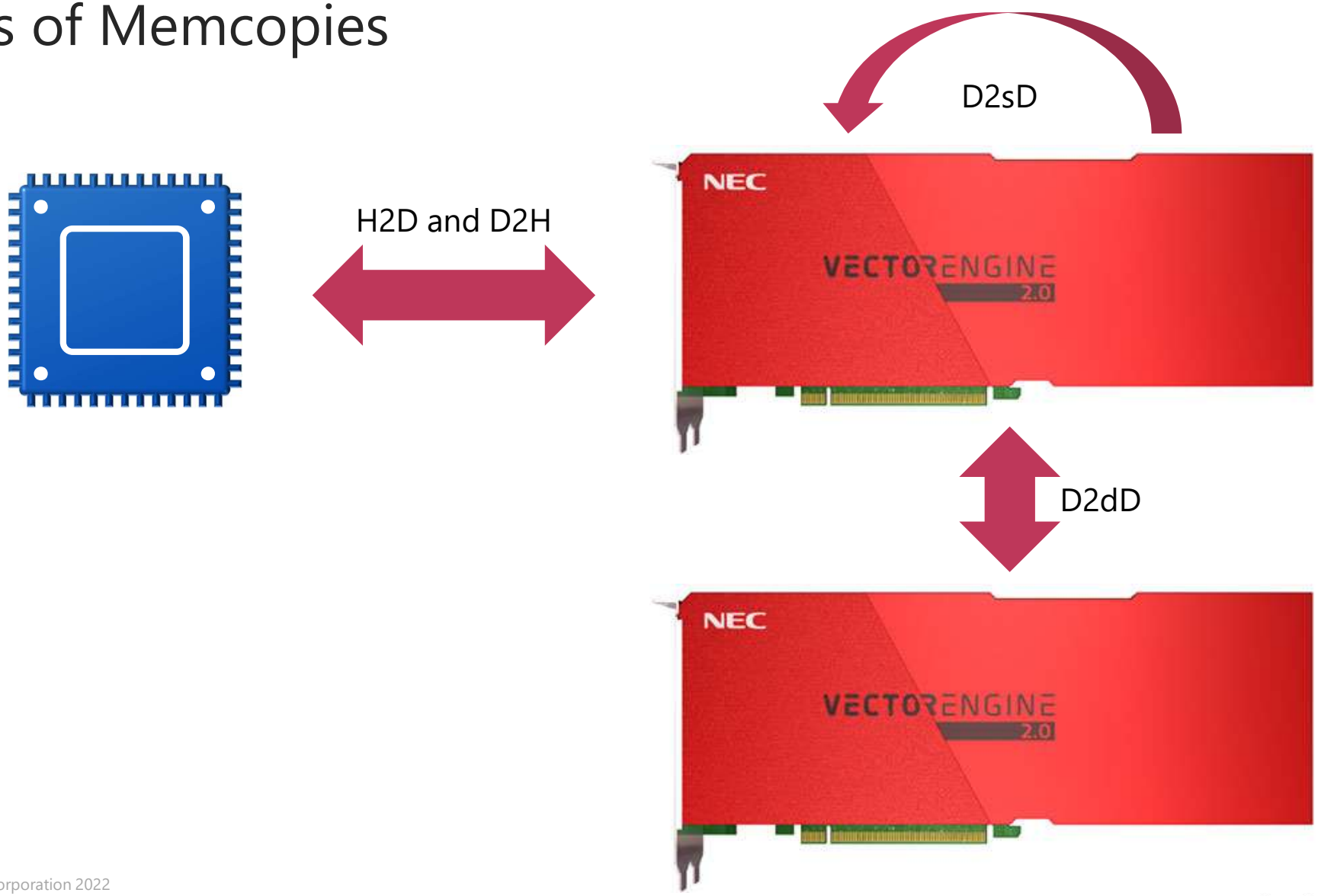
\Orchestrating a brighter world NEC

More details in our SX-Aurora Article:

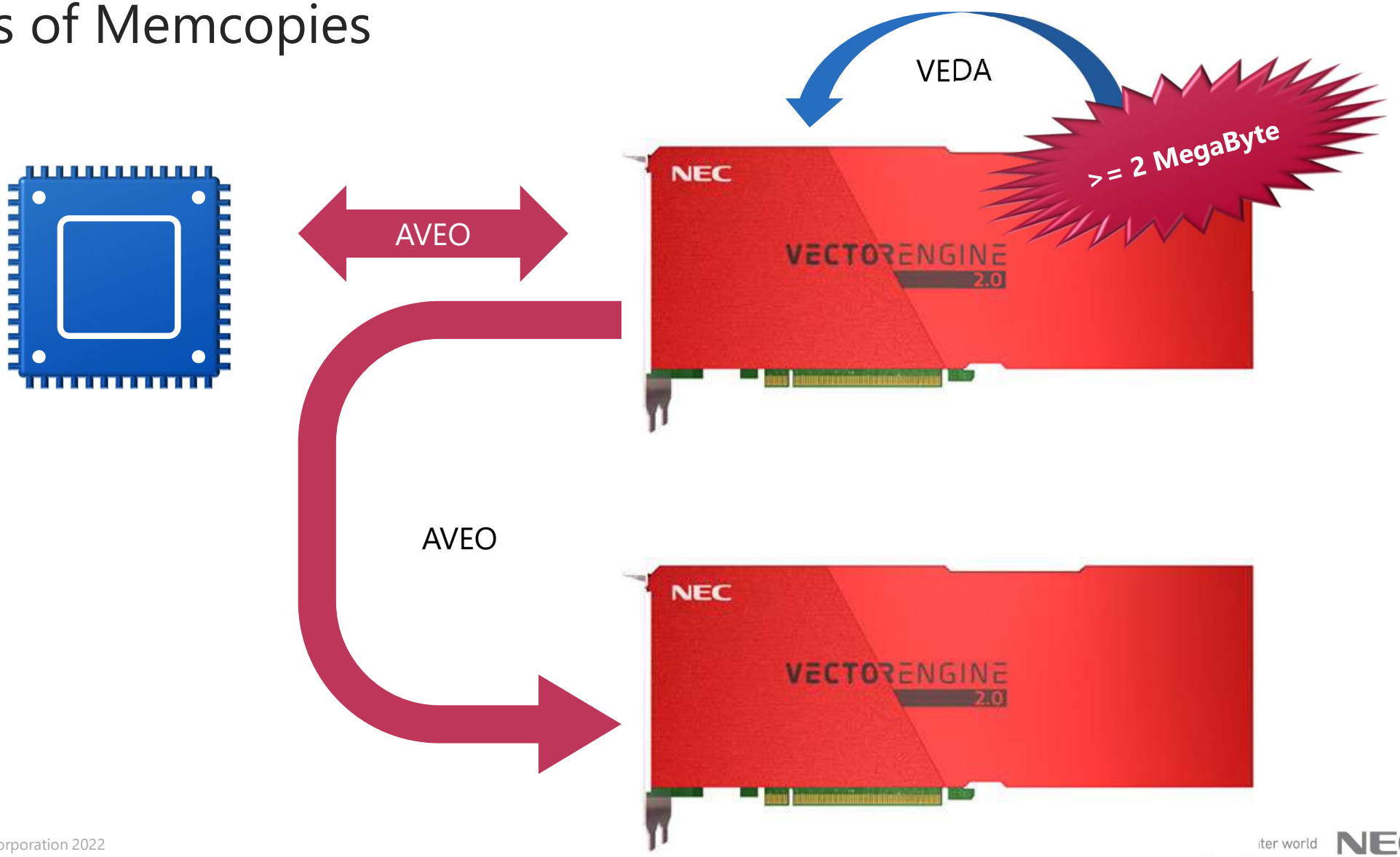
VEDA: Best practices to use hybrid programming on the NEC SX-Aurora TSUBASA

Improving performance of memcopy operations

Types of Memcopies



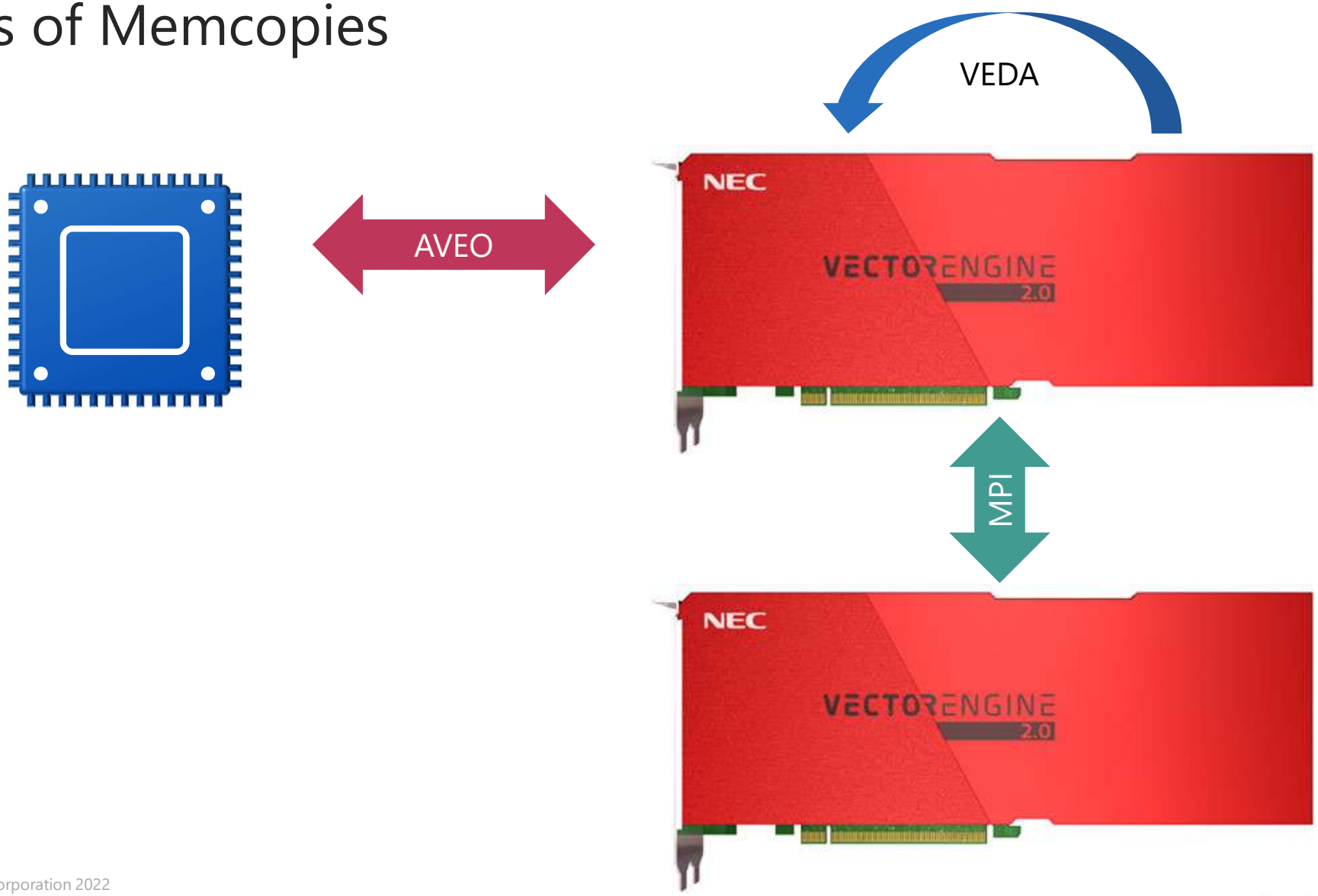
Types of Memcopies



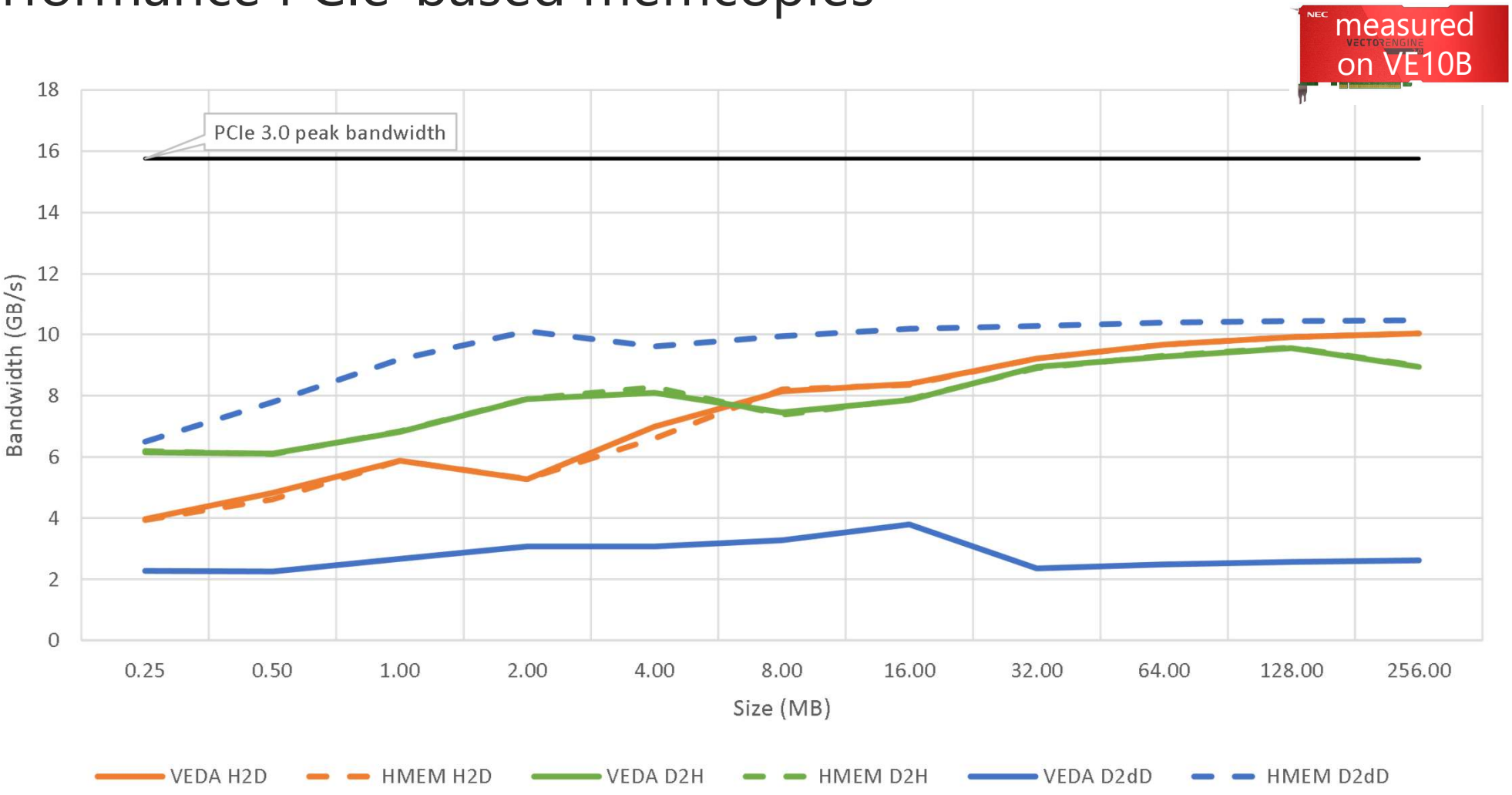
NEC MPI provides direct memcopy path

```
// VEDAhmemptr API -------------------------------------------------------------
VEDAresult      vedaArgsSetHMEM         (VEDAargs args, const int idx, const VEDAhmemptr value);
VEDAresult      vedaHMemAlloc           (VEDAhmemptr* ptr, size_t size);
VEDAresult      vedaHMemFree            (VEDAhmemptr ptr);
VEDAresult      vedaHMemPtr             (void** ptr, VEDAhmemptr hptr);
VEDAresult      vedaHMemcpy             (void* dst, void* src, size_t ByteCount);
VEDAresult      vedaHMemcpyDtoX         (void* dst, VEDAdeviceptr src, size_t ByteCount);
VEDAresult      vedaHMemcpyDtoXAsync    (void* dst, VEDAdeviceptr src, size_t ByteCount, VEDAstream stream);
VEDAresult      vedaHMemcpyXtoD         (VEDAdeviceptr dst, void* src, size_t ByteCount);
VEDAresult      vedaHMemcpyXtoDAsync    (VEDAdeviceptr dst, void* src, size_t ByteCount, VEDAstream stream);
VEDAresult      vedaHMemsetD128         (VEDAhmemptr dstDevice, uint64_t x, uint64_t y, size_t N);
VEDAresult      vedaHMemsetD128Async    (VEDAhmemptr dstDevice, uint64_t x, uint64_t y, size_t N, VEDAstream hStream);
VEDAresult      vedaHMemsetD16          (VEDAhmemptr dstDevice, uint16_t value, size_t N);
VEDAresult      vedaHMemsetD16Async     (VEDAhmemptr dstDevice, uint16_t value, size_t N, VEDAstream hStream);
VEDAresult      vedaHMemsetD2D128       (VEDAhmemptr dstDevice, size_t dstPitch, uint64_t x, uint64_t y, size_t Width, size_t Height);
VEDAresult      vedaHMemsetD2D128Async  (VEDAhmemptr dstDevice, size_t dstPitch, uint64_t x, uint64_t y, size_t Width, size_t Height, VEDAstream hStream);
VEDAresult      vedaHMemsetD2D16        (VEDAhmemptr dstDevice, size_t dstPitch, uint16_t value, size_t Width, size_t Height);
VEDAresult      vedaHMemsetD2D16Async   (VEDAhmemptr dstDevice, size_t dstPitch, uint16_t value, size_t Width, size_t Height, VEDAstream hStream);
VEDAresult      vedaHMemsetD2D32        (VEDAhmemptr dstDevice, size_t dstPitch, uint32_t value, size_t Width, size_t Height);
VEDAresult      vedaHMemsetD2D32Async   (VEDAhmemptr dstDevice, size_t dstPitch, uint32_t value, size_t Width, size_t Height, VEDAstream hStream);
VEDAresult      vedaHMemsetD2D64        (VEDAhmemptr dstDevice, size_t dstPitch, uint64_t value, size_t Width, size_t Height);
VEDAresult      vedaHMemsetD2D64Async   (VEDAhmemptr dstDevice, size_t dstPitch, uint64_t value, size_t Width, size_t Height, VEDAstream hStream);
VEDAresult      vedaHMemsetD2D8         (VEDAhmemptr dstDevice, size_t dstPitch, uint8_t value, size_t Width, size_t Height);
VEDAresult      vedaHMemsetD2D8Async    (VEDAhmemptr dstDevice, size_t dstPitch, uint8_t value, size_t Width, size_t Height, VEDAstream hStream);
VEDAresult      vedaHMemsetD32          (VEDAhmemptr dstDevice, uint32_t value, size_t N);
VEDAresult      vedaHMemsetD32Async     (VEDAhmemptr dstDevice, uint32_t value, size_t N, VEDAstream hStream);
VEDAresult      vedaHMemsetD64          (VEDAhmemptr dstDevice, uint64_t value, size_t N);
VEDAresult      vedaHMemsetD64Async     (VEDAhmemptr dstDevice, uint64_t value, size_t N, VEDAstream hStream);
VEDAresult      vedaHMemsetD8           (VEDAhmemptr dstDevice, uint8_t value, size_t N);
VEDAresult      vedaHMemsetD8Async      (VEDAhmemptr dstDevice, uint8_t value, size_t N, VEDAstream hStream);
```

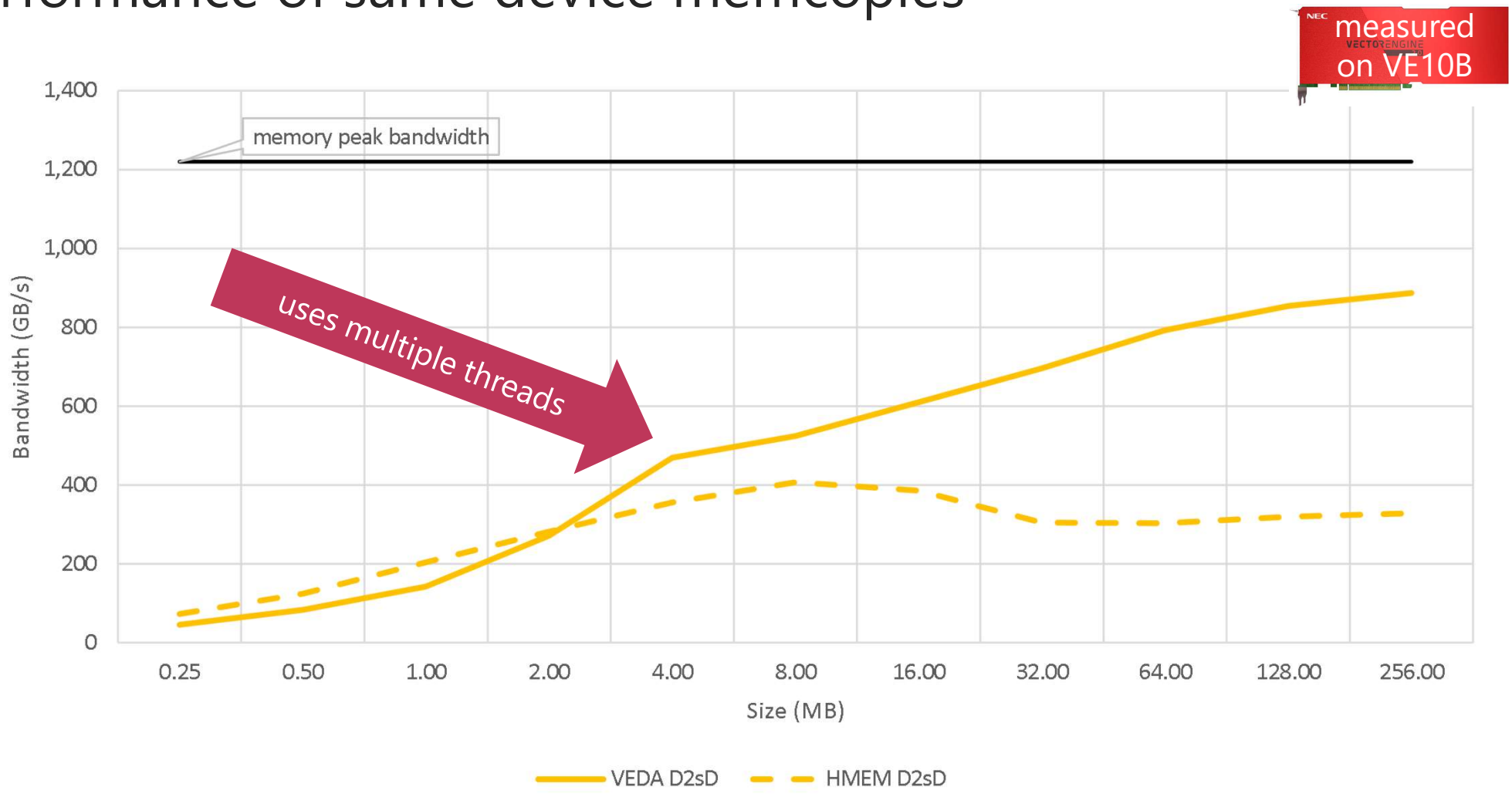
Types of Memcopies



Performance PCIe-based memcopies



Performance of same device memcopies



When to use **VEDAdeviceptr** or **VEDAhmemptr**?

- **VEDAdeviceptr**
  - Optimized for asynchronous executions on a single device
  - *Use whenever data only stays on a single device*
- **VEDAhmemptr**
  - Optimized for cross-device data sharing
  - *Use whenever data needs to be shared with other devices*

\Orchestrating a brighter world NEC

More details in our SX-Aurora Article:

VEDA: Best practices to use hybrid programming on the NEC SX-Aurora TSUBASA

New VEDA C++ API

- C++ and templated function fetching
- Buffer based memory model with automatic garbage collection
- “Futures” to fetch return values of async function calls

VEDA Profiling API

- Allows to register a callback function to get informed when operations get executed or have been completed
- **Events:**
  - (H)MemAlloc
  - (H)MemFree
  - (H)Memcpy (H2D, D2H, D2D)
  - Device/Host Kernel calls

NEC Laboratories Europe, SX-Aurora Articles

\Orchestrating a brighter world NEC

VEDA: Best practices to use hybrid programming on the NEC SX-Aurora TSUBASA
Nicolas Weber, NEC Laboratories Europe

SOL: Reducing the Maintenance Overhead for Integrating Hardware Support into AI Frameworks
Nicolas Weber, NEC Laboratories Europe

AVEO-VEDA: Hybrid Programming for the NEC Vector Engine
Nicolas Weber, NEC Laboratories Europe
Erich Focht, NEC Deutschland HPCE

>> https://www.nec.com/en/global/solutions/hpc/articles/ <<

NEC Laboratories Europe, Open Positions

\Orchestrating a brighter world NEC

>> https://neclab.eu/jobs <<

\Orchestrating a brighter world NEC

NEC Laboratories Europe research and develop cutting-edge technology to create innovative social solutions. Researchers choose their projects, lead scientific discovery and are consistently published in top, peer-reviewed scientific papers. NEC Laboratories Europe technology promotes NEC Group solutions for a better society in the fields of digital health, communication infrastructure, safer cities and public services. For more information visit https://www.neclab.eu.

NEC Laboratories Europe is looking for a

Visiting Researcher on AI Compilers

[ref: 2210-455-ISS]

NEC Laboratories Europe is looking for a Visiting Researcher (Ph.D. student, or postdoc) to conduct individual research in the area of optimizing AI compilers.

In this visiting period you will work in close collaboration with the NEC SOL AI compiler team. The goal of this project is to implement and evaluate methods to reduce memory consumption during AI training.

In particular, you will explore trade-offs in memorization vs re-computation of intermediate results during AI training. The activity will include implementation and experimental evaluation of your work (performance, energy consumption and memory), and, if needed, research in methods to search optimal solutions when in presence of multiple options.
Depending of your progress and results, we expect this activity to contribute to scientific papers and IPR generation.

We seek people with individual creativity as well as a strong teamwork attitude, with a strong desire to work on cutting edge technology in one of the top research labs in Europe. Our working language is English. The duration of visiting period is initially limited to 4-6 months.

NEC Laboratories Europe is located in Heidelberg, one of Germany's most beautiful and international cities. The city and the surrounding Rhein-Neckar region offers many exciting cultural and outdoor activities.

Home office might be possible in later stages of the visiting periods, but the onboarding phase requires physical presence in the office.

Desired Skills and Experience:

- Ability to work in multi-cultural teams with a positive, can-do attitude is a strong requirement.
- Excellent written and spoken communication skills
- Solid C++ knowledge (C++11 and above) is a must have
- Strong proficiency in Python and Python-based AI frameworks such as PyTorch, or TensorFlow.

The following skills are a plus:

- Knowledge of AI computations and automatic differentiation are a plus.
- A track record of related scientific publications is a plus.

We are looking for individuals with a broad background in computer science. A masters in Computer Science, Computational Mathematics or related field is required.

Please send your application **October 31, 2022** electronically via the applications web system https://www.neclab.eu/join-us with reference to **[2210-455-ISS].**

For questions, please contact:
Saverio Niccolini / General Manager
Tel. +49-6221-4342-0 /

NEC Laboratories Europe GmbH / Kurfürsten-Anlage 36 / 69115 Heidelberg / Germany / www.neclab.eu

\Orchestrating a brighter world NEC

NEC Laboratories Europe research and develop cutting-edge technology to create innovative social solutions. Researchers choose their projects, lead scientific discovery and are consistently published in top, peer-reviewed scientific papers. NEC Laboratories Europe technology promotes NEC Group solutions for a better society in the fields of digital health, communication infrastructure, safer cities and public services. For more information visit https://www.neclab.eu.

NEC Laboratories Europe has an immediate opening for a

Visiting Researcher on "Impact of inaccurate computations during AI training"

[ref: 2210-456-ISS]

NEC Laboratories Europe is looking for a Visiting Researcher (Ph.D. student, or postdoc) to conduct individual research in the area of AI frameworks.

In this internship you will work in close collaboration with the NEC SOL AI compiler team. You will investigate the impact of false and inaccurate computations during AI trainings. The goal of this project is to understand how such errors affect neural networks and how this might have impacted the results of papers that relied on the correctness of computations in modern AI frameworks.

You will implement, train and evaluate different neural networks architectures and analyze how this kind of error affects the accuracy, as well as any other relevant metrics. Depending of your progress and results, we expect this activitiy to conribute to scientific papers.

We seek people with individual creativity as well as a strong teamwork mentality. Our working language is English. The duration of the visiting period is initially limited to 4-6 months.

NEC Laboratories Europe is located in Heidelberg, one of Germany's most beautiful cities. The city and the surrounding Rhein-Neckar region offers many exciting cultural and outdoor activities.

Home office might be possible in later stages of the internship, but the onboarding phase requires physical presence in the office.

Desired Skills and Experience:

- Ability to work in multi-cultural teams with a positive, can-do attitude is a strong requirement.
- Excellent written and spoken communication skills
- Solid knowledge in developing AI training pipelines.
- Strong proficiency in Python and Python-based AI frameworks such as PyTorch, or TensorFlow.

The following skills are a plus

- Knowledge of AI computations and automatic differentiation are a plus.
- A track record of related scientific publications is a plus.

We are looking for individuals with a broad background in computer science. A masters in Computer Science, Computational Mathematics or related field is required.

Please send your application **October 31, 2022** electronically via the applications web system **https://www.neclab.eu/join-us** with reference to **[2210-456-ISS].**

For questions, please contact:
Saverio Niccolini / General Manager
Tel. +49-6221-4342-0 /

NEC Laboratories Europe GmbH / Kurfürsten-Anlage 36 / 69115 Heidelberg / Germany / www.neclab.eu