oneAPI mission: enabling us all to benefit from Heterogeneous Computing

James Reinders, engineer
June 16, 2022
Heterogeneous Systems – Programming them

My talk today:

1. Heterogeneous Systems are here to stay and will be ubiquitous (like parallelism)

2. Look at the essentials of SYCL* (this is just for C++)

3. oneAPI: open, multivendor, multiarchitecture support that spans programming languages

4. This is OUR problem – let’s solve it together
2022: 25th Anniversary ASCI Red supercomputer takes #1 spot

#1 system for seven Top 500 lists (still a record) - from June 1997 through June 2000

- First TeraFLOP/s computer in the world.
- 7264 processors (cores) of Intel® Pentium® Pro processors @200MHz for 1.45 TeraFLOP/s. Later upgraded to 9632 Pentium II Over-Drive processors @333MHz for 3.21 TeraFLOP/s.
- Parallel programming focused on distributed parallelism (message passing)
2022: 25th Anniversary ASCI Red supercomputer takes #1 spot

#1 system for eight Top 500 lists (still a record) - from June 1997 through June 2000

- First TeraFLOP/s computer in the world.
- 7264 processors (cores) of Intel® Pentium® Pro processors @200MHz for 1.45 TeraFLOP/s. Later upgraded to 9632 Pentium II Over-Drive processors @333MHz for 3.21 TeraFLOP/s.
- Parallel programming focused on distributed parallelism (message passing)

What has happened in the 25 years since?

- “nodes” have become much fatter
  - Multicore, multisocket, and heterogeneous compute
- nodes require parallel programming of all kinds – distributed, share memory, offload
My talk today:

1. Heterogeneous Systems are here to stay and will be ubiquitous (like parallelism)

2. Look at the essentials of SYCL* (this is just for C++)

3. oneAPI:
   open, multivendor, multiarchitecture support that spans programming languages

4. This is OUR problem – let’s solve it together
Our quest for more performance is eternal; how we obtain it adapts to the times

Source: tinyurl.com/karlruppdata (CC BY 4.0 license)
Our quest for more performance is eternal; how we obtain it adapts to the times

Source: tinyurl.com/karlruppdata (CC BY 4.0 license)
Our quest for more performance is eternal; how we obtain it adapts to the times.

Chiplets can allow this to continue even when if node feature sizes do not.
Computer trends: Parallel and Heterogeneous

Why Parallel?
Desire to get more work done, by having more workers.

Workers = compute units, devices, processing units, etc. (e.g., CPU, GPU, FPGA, ASIC, AI chip)
Computer trends: Parallel and Heterogeneous

Why Parallel?
Desire to get more work done, by having more workers.

Why Heterogeneous?
Desire to get more work done, by having different types of workers.

*Workers = compute units, devices, processing units, etc. (e.g., CPU, GPU, FPGA, ASIC, AI chip)*
Computer trends: Parallel and Heterogeneous

Why Parallel?
Desire to get more work done, by having more workers.

Why Heterogeneous?
Desire to get more work done, by having different types of workers. And... well planned specialization can be more power efficient.

Workers = compute units, devices, processing units, etc.
(e.g., CPU, GPU, FPGA, ASIC, AI chip)
A New Golden Age for Computer Architecture

“The next decade will see a **Cambrian explosion of novel computer architectures**, meaning exciting times for computer architects in academia and industry.”

ACM Turing Award laureates John Hennessy and David Patterson (CACM, Feb 2019, Vol 62, No 2, pp 48-60)

[https://tinyurl.com/HPcambrian](https://tinyurl.com/HPcambrian) <<< HIGHLY RECOMMENDED READING
A New Golden Age for Computer Architecture

“The next decade will see a **Cambrian explosion of novel computer architectures**, meaning exciting times for computer architects in academia and industry.”

ACM Turing Award laureates John Hennessy and David Patterson (CACM, Feb 2019, Vol 62, No 2, pp 48-60)

[https://tinyurl.com/HPcambrian](https://tinyurl.com/HPcambrian)

---

**Products**

- Novel On-Die Accelerators
- GPU/Data Parallel
- Spatial/Dataflow
- Deep Learning Optimized
- Blockchain
- Mix & Match Nearly Endless Combinations

---

**Research**

- Neuromorphic
- Graph Analytics and More...

HPSC 2022
Observation

• When a computer was homogeneous – we could program it with any tool, even if it was unique or proprietary.

• When a computer is heterogeneous – we need tools to work together.
Before heterogeneous systems

My application

compiler
(libraries & tools too)

CPU

Portability was a function of the language used.

C, C++, Fortran, Java*, Python*

I didn’t care if the compiler, etc., was proprietary or not – since the target system was single vendor, single architecture.

CPUs aimed to be excellent at the same things
Observation

- When a computer was homogeneous – we could program it with any tool, even if it was unique or proprietary.
- When a computer is heterogeneous – we need tools to work together.
Now, with heterogeneous systems

My application → compiler (libraries & tools too) → CPU

open, multivendor, multiarchitecture vs. walled-garden

matters like it never has

The more XPU (device) the world gets, the more this matters.
Portability is not enough in a heterogeneous world.

Performance Portability - Definition and Metric:

“A measurement of an application’s performance efficiency for a given problem that can be executed correctly on all platforms in a given set.”

\[ PP(a, p, H) = \begin{cases} \frac{|H|}{1} & \text{if } i \text{ is supported } \forall i \in H \\ \sum_{i \in H} e_i(a, p) & \text{otherwise} \end{cases} \]

Anything portable is “performance portable”. The question becomes: “How performance portable is it?”

- Yes/No answer for “is it PP?”
- Captures “average” performance in \( H \)
- Architectural and Application Efficiency

**Recommended reading:**
Navigating Performance, Portability and Productivity
https://tinyurl.com/NavigatePerf

Can we survive the diversity?

*Do we have a choice?*

make it much easier with “open, multivendor, multiarchitecture”
A List of the...

Many Real-World Challenges for Effective Programming of Heterogeneous Systems

• be open, multivendor, and multiarchitecture – always
  • Pass three tests:
    1. Freedom to use any device (regardless of vendor or architecture)
    2. Ability to access maximum performance
    3. A future for my investments in coding

• support across many programming languages
• performance portability
• commonality for developers
• commonality under the covers
Heterogeneous Systems – Programming them

My talk today:

1. Heterogeneous Systems are here to stay and will be ubiquitous (like parallelism)

2. Look at the essentials of SYCL* (this is just for C++)

3. oneAPI:
   open, multivendor, multiarchitecture support that spans programming languages

4. This is OUR problem – let’s solve it together
```cpp
#include <CL/sycl.hpp>
#include <iostream>

int main() {
    sycl::queue Q;
    std::cout << "Running on: " << Q.get_device().get_info<sycl::info::device::name>() << std::endl;

    int sum;
    std::vector<int> data{1, 1, 1, 1, 1, 1, 1};

    sycl::buffer<int> sum_buf(&sum, 1);
    sycl::buffer<int> data_buf(data);

    Q.submit([&](sycl::handler& h) {
        sycl::accessor buf_acc{data_buf, h, read_only};

        h.parallel_for(sycl::range<1>{8},
                       sycl::reduction(sum_buf, h, std::plus<>()),
                       [=](sycl::id<1> idx, auto& sum)
                       {
                           sum += buf_acc[idx];
                       });
    });

    sycl::host_accessor result{sum_buf, read_only};
    std::cout << "Sum equals " << result[0] << std::endl;

    return 0;
}
```
```cpp
#include <CL/sycl.hpp>
#include <iostream>

int main() {
    sycl::queue Q;
    std::cout << "Running on: " << Q.get_device().get_info<sycl::info::device::name>() << std::endl;

    int sum;
    std::vector<int> data{1, 1, 1, 1, 1, 1, 1, 1};

    sycl::buffer<int> sum_buf(&sum, 1);
    sycl::buffer<int> data_buf(data);

    Q.submit([&](sycl::handler& h) {
        sycl::accessor buf_acc{data_buf, h, sycl::access::read_only};

        h.parallel_for(sycl::range<1>{8},
                       sycl::reduction(sum_buf, h, std::plus<>()),
                       [=](sycl::id<1> idx, auto& sum) {
                           sum += buf_acc[idx];
                       });
    });

    sycl::host_accessor result{sum_buf, sycl::access::read_only};
    std::cout << "Sum equals " << result[0] << std::endl;

    return 0;
}
```
```cpp
#include <CL/sycl.hpp>
#include <iostream>

int main() {
    sycl::queue Q;
    std::cout << "Running on: " << Q.get_device().get_info<sycl::info::device::name>() << std::endl;

    int sum;
    std::vector<int> data{1, 1, 1, 1, 1, 1, 1, 1};

    sycl::buffer<int> sum_buf(&sum, 1);
    sycl::buffer<int> data_buf(data);

    Q.submit([&](sycl::handler& h) {
        sycl::accessor buf_acc{data_buf, h, read_only};

        h.parallel_for(sycl::range<1>{8},
            sycl::reduction(sum_buf, h, std::plus<>()),
            [=](sycl::id<1> idx, auto& sum)
        {
            sum += buf_acc[idx];
        });
    });

    sycl::host_accessor result(sum_buf, read_only);
    std::cout << "Sum equals " << result[0] << std::endl;

    return 0;
}
```
```cpp
#include <CL/sycl.hpp>
#include <iostream>

int main() {
    sycl::queue Q;
    std::cout << "Running on: " << Q.get_device().get_info<sycl::info::device::name>() << std::endl;

    int sum;
    std::vector<int> data{1, 1, 1, 1, 1, 1, 1, 1};

    sycl::buffer<int> sum_buf(sum, 1);
    sycl::buffer<int> data_buf(data);
    Q.submit([&](sycl::handler& h) {
        sycl::accessor buf_acc[data_buf, h, read_only];

        h.parallel_for(sycl::range<1>{8},
                        sycl::reduction(sum_buf, h, std::plus<>()),
                        [=](sycl::id<1> idx, auto& sum)
                        {
                            sum += buf_acc[idx];
                        });
    });

    sycl::host_accessor result(sum_buf, read_only);
    std::cout << "Sum equals " << result[0] << std::endl;

    return 0;
}
```
SYCL* is expressive & exposes control

- Device queries
- Queue & context control
- OpenCL*-like buffers and unified shared memory
- Optional asynchrony & task DAG
- Generic groups & groups algorithms
- SPMD-to-SIMD interoperability (InvokeSIMD)
- JIT & Specialization Constants
- Interoperability with OpenMP

Book (PDF) Download: tinyurl.com/DataParallelCpp
Heterogeneous Systems – Programming them

My talk today:

1. Heterogeneous Systems are here to stay and will be ubiquitous (like parallelism)

2. Look at the essentials of SYCL* (this is just for C++)

3. oneAPI: open, multivendor, multiarchitecture support that spans programming languages

4. This is OUR problem – let’s solve it together
C++ programming is just one piece

• LIBRARIES are KEY

• Supporting MANY languages is IMPORTANT
e.g., Fortran, Python*, C, Julia*, OpenMP*, MPI

oneAPI tools support all our needs
oneAPI: One Name, Two Distinct Objectives

- Open industry specification
- Open-source repo and development
- Community driven
- Multivendor implementations

- Intel’s implementation
- Toolkits optimized for Intel® hardware
- Free to download and use

• Standard C++ with SYCL
• Standardized interfaces for common libraries
• Standardized hardware interface

oneAPI Industry Specification

Direct Programming

API-Based Programming

Low-Level Hardware Interface (Level Zero)

Libraries
- Math
- Threading
- Parallel STL
- Ray Tracing
- Analytics/ML
- DNN
- ML Comm
- Volumetric Rendering
- Video Processing
- Signal Processing
- Image Processing
- Image Denoise

oneAPI: One Name, Two Distinct Objectives

HPSC 2022
An open specification and initiative to standardize programming of accelerated processing units

oneapi.io

Intel’s product implementation of the oneAPI specification free

software.intel.com/oneAPI
Amazing already, and lots of interesting work and research remain.
Common “under the covers” - lots of work to do!

Composability It’s important. 😊

Heterogeneous is leading to mix-and-match like nothing before, therefore... composability matters even more.
Heterogeneous Systems – Programming them

My talk today:

1. Heterogeneous Systems are here to stay and will be ubiquitous (like parallelism)

2. Look at the essentials of SYCL* (this is just for C++)

3. oneAPI:
   open, multivendor, multiarchitecture support that spans programming languages

4. This is OUR problem – let’s solve it together
oneAPI is important

We do STRESS our belief in the need to bring us all together to create an open, multivendor, multiarchitecture, multilanguage future
A List of the...

Many Real-World Challenges for Effective Programming of Heterogeneous Systems

• be open, multivendor, and multiarchitecture – always
  • Pass three tests:
    1. Freedom to use any device (regardless of vendor or architecture)
    2. Ability to access maximum performance
    3. A future for my investments in coding

• support across many programming languages

• performance portability

• commonality for developers

• commonality under the covers
It’s a Journey

We started oneAPI with a good idea and a great implementation

We knew enough to propose initial specifications

We are rapidly iterating and refining through community feedback

oneAPI has evolved
It’s a Journey

We started oneAPI with a good idea and a great implementation.

We knew enough to propose initial specifications.

We are rapidly iterating and refining through community feedback.

oneAPI has evolved.

Much work remains – join us – with oneAPI in creating an open, multivendor, multiarchitecture, multilanguage future.

https://oneapi.io
https://software.intel.com/oneAPI
Thank you

We started oneAPI with a good idea

We knew enough to propose initial specifications

We are rapidly iterating and refining through community feedback

oneAPI has evolved

Much work remains – join us – with oneAPI - in creating an open, multivendor, multiarchitecture, multilanguage future

Have a GREAT conference!

https://oneapi.io
https://software.intel.com/oneAPI
Disclaimers & Notices

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Intel technologies’ features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No product or component can be absolutely secure. Check with your system manufacturer or retailer or learn more at intel.com.

Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others. © Intel Corporation

Khronos® is a registered trademark and SYCL™ and SPIR™ are trademarks of The Khronos Group Inc. OpenCL™ is a trademark of Apple Inc. used by permission by Khronos.