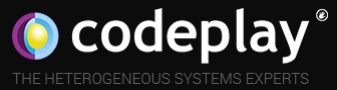


The landscape of Parallel Programming Models
Part 1: Performance, Portability & Productivity

Michael Wong
Codeplay Software Ltd.
Distinguished Engineer

IXPUG 2020

Products

Acoran

Integrates all the industry standard technologies needed to support a very wide range of AI and HPC

ComputeCpp

C++ platform via the SYCL™ open standard, enabling vision & machine learning e.g. TensorFlow™

ComputeAorta

The heart of Codeplay's compute technology enabling OpenCL™, SPIR-V™, HSA™ and Vulkan™

Markets

High Performance Compute (HPC)
Automotive ADAS, IoT, Cloud Compute
Smartphones & Tablets
Medical & Industrial

Technologies: Artificial Intelligence
Vision Processing
Machine Learning
Big Data Compute

Company

Leaders in enabling high-performance software solutions for new AI processing systems

Enabling the toughest processors with tools and middleware based on open standards

Established 2002 in Scotland with ~80 employees

Customers

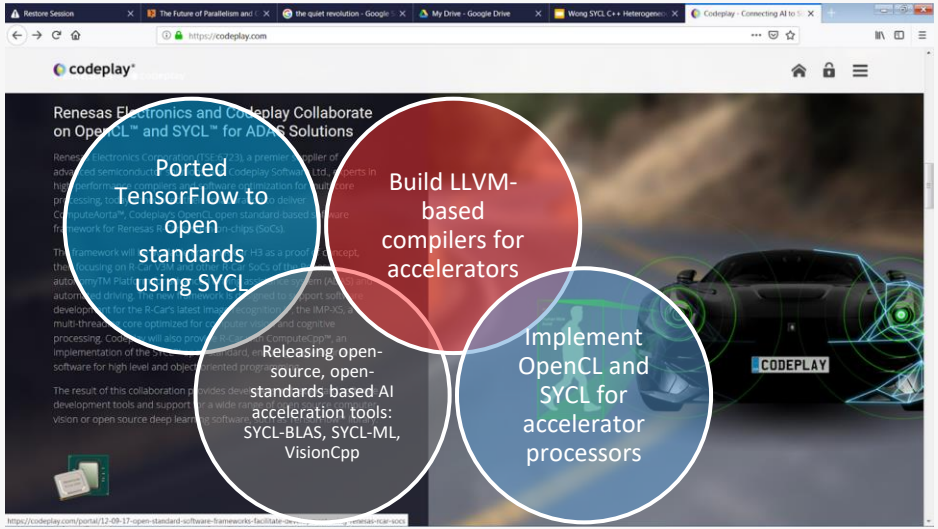
arm
BROADCOM
CEVA
Imagination
intel
RENESAS
SYNOPSYS

And many more!

Distinguished Engineer

Michael Wong

- Chair of SYCL Heterogeneous Programming Language
- C++ Directions Group
- ISOCPP.org Director, VP
  http://isocpp.org/wiki/faq/wg21#michael-wong
- michael@codeplay.com
- fraggamuffin@gmail.com
- Head of Delegation for C++ Standard for Canada
- Chair of Programming Languages for Standards Council of Canada
  Chair of WG21 SG19 Machine Learning
  Chair of WG21 SG14 Games Dev/Low Latency/Financial Trading/Embedded
- Editor: C++ SG5 Transactional Memory Technical Specification
- Editor: C++ SG1 Concurrency Technical Specification
- MISRA C++ and AUTOSAR
- Chair of Standards Council Canada TC22/SC32 Electrical and electronic components (SOTIF)
- Chair of UL4600 Object Tracking
- http://wongmichael.com/about
- C++11 book in Chinese:
  https://www.amazon.cn/dp/B00ETOV2OQ



We build GPU compilers for semiconductor companies
Now working to make AI/ML heterogeneous acceleration safe for autonomous vehicle

Acknowledgement and Disclaimer

THIS WORK REPRESENTS THE VIEW OF THE AUTHOR AND DOES NOT NECESSARILY REPRESENT THE VIEW OF CODEPLAY.

OTHER COMPANY, PRODUCT, AND SERVICE NAMES MAY BE TRADEMARKS OR SERVICE MARKS OF OTHERS.

Numerous people internal and external to the original C++/Khronos group/OpenMP, in industry and academia, have made contributions, influenced ideas, written part of this presentations, and offered feedbacks to form part of this talk. These in clude Bjarne Stroustrup, Joe Hummel, Botond Ballo, Simon Mcintosh-Smith, as well as many others.

But I claim all credit for errors, and stupid mistakes. **These are mine, all mine! You can't have them.**

Legal Disclaimer

THIS WORK REPRESENTS THE VIEW OF THE AUTHOR AND DOES NOT NECESSARILY REPRESENT THE VIEW OF CODEPLAY.

OTHER COMPANY, PRODUCT, AND SERVICE NAMES MAY BE TRADEMARKS OR SERVICE MARKS OF OTHERS.

Disclaimers

NVIDIA, the NVIDIA logo and CUDA are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and/or other countries

Codeplay is not associated with NVIDIA for this work and it is purely using public documentation and widely available code

3 Act Play

1. Performance, Portability, Productivity
2. The four horsemen of heterogeneous programming
3. C++, OpenCL, OpenMP, SYCL

Act 1

Performance, Portability, Productivity

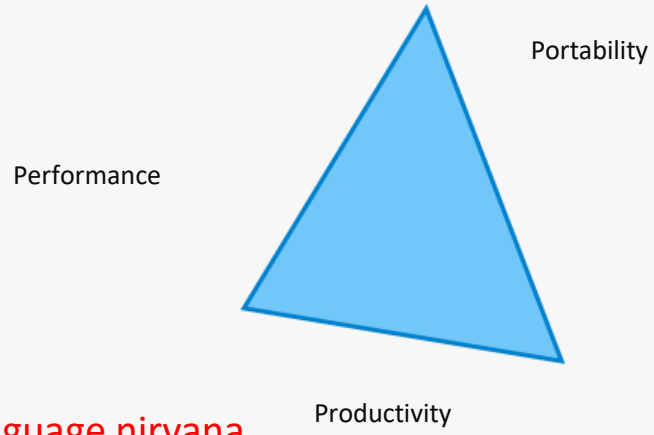
Isn’t Parallel/Concurrent/Heterogeneous Programming Hard?

Houston, we have a software crisis!

So What are the Goals?

Goals of Parallel Programming over and above sequential programming

1. Performance
2. Productivity
3. Portability



The Iron Triangle of Parallel Programming language nirvana

Why does productivity make the list?

Performance

- Broadly includes scalability and efficiency
- If not for performance why not just write sequential program?
- parallel programming is primarily a performance optimization, and, as such, it is one potential optimization of many.

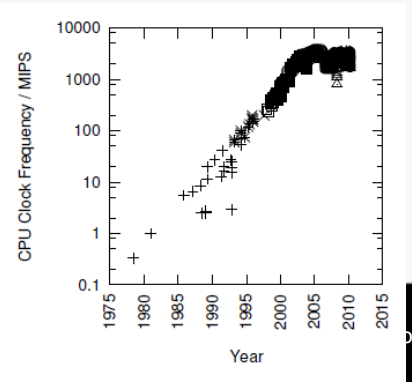
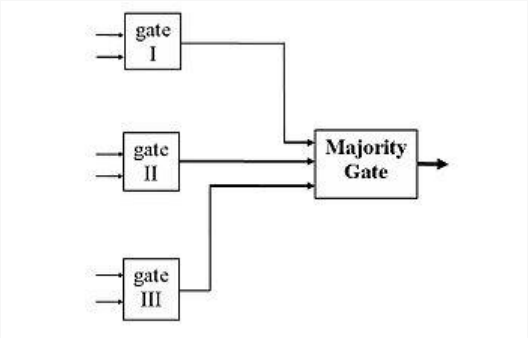


Diagram thanks to Paul McKenney

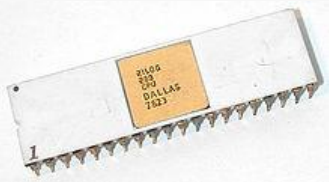
Are there no cases where parallel programming is about something other than performance?

AUTOSAR



Productivity

Perhaps at one time, the sole purpose of parallel software was performance. Now, however, productivity is gaining the spotlight.



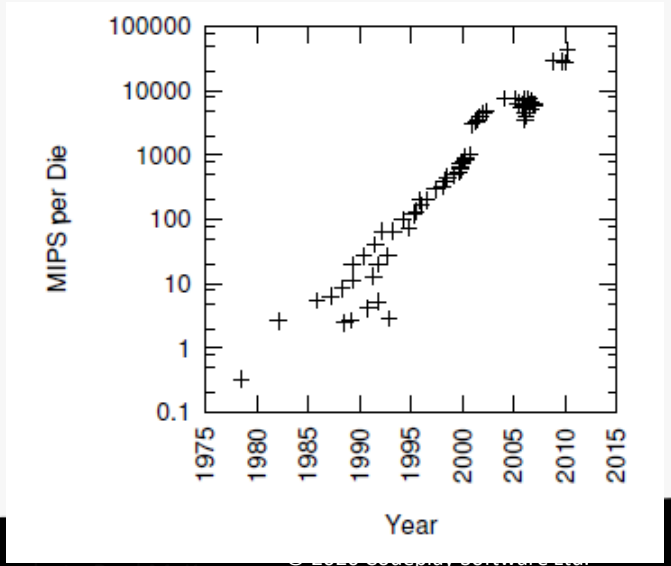
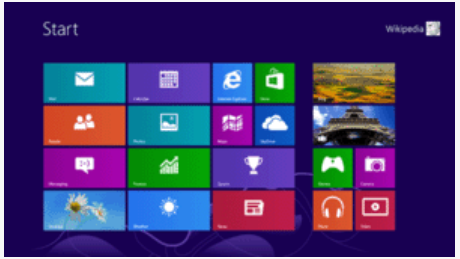


Diagram thanks to Paul McKenney



Given how cheap parallel systems have become, how can anyone afford to pay people to program them?

Iron Triangle of Parallel Programming Language Nirvana

Portability

Performance

Productivity

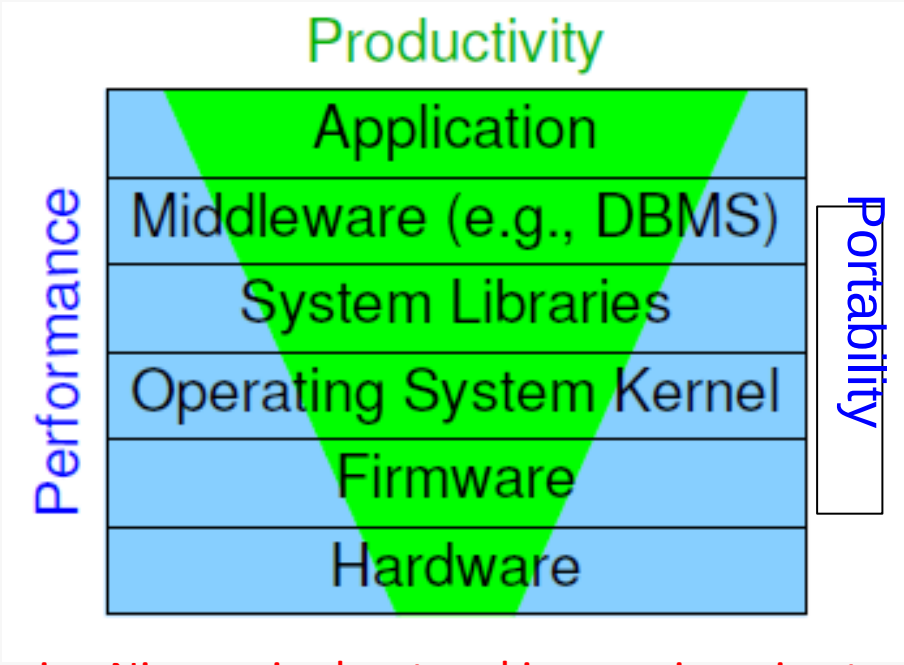
Performance Portability Productivity

OpenCL

OpenMP

CUDA

SYCL



Iron Triangle of Parallel Programming Nirvana is about making engineering tradeoffs

Concurrency vs Parallelism

What makes parallel or concurrent programming harder than serial programming? What's the difference? How much of this is simply a new mindset one has to adopt?

Which one to choose?

OpenACC®
DIRECTIVES FOR ACCELERATORS

OpenMP™

CILK ARTS

SYCL™

C++ AMP
Accelerated Massive Parallelism with Microsoft Visual C++

PPL
stands for
Parallel Patterns Library
Abbreviations.com

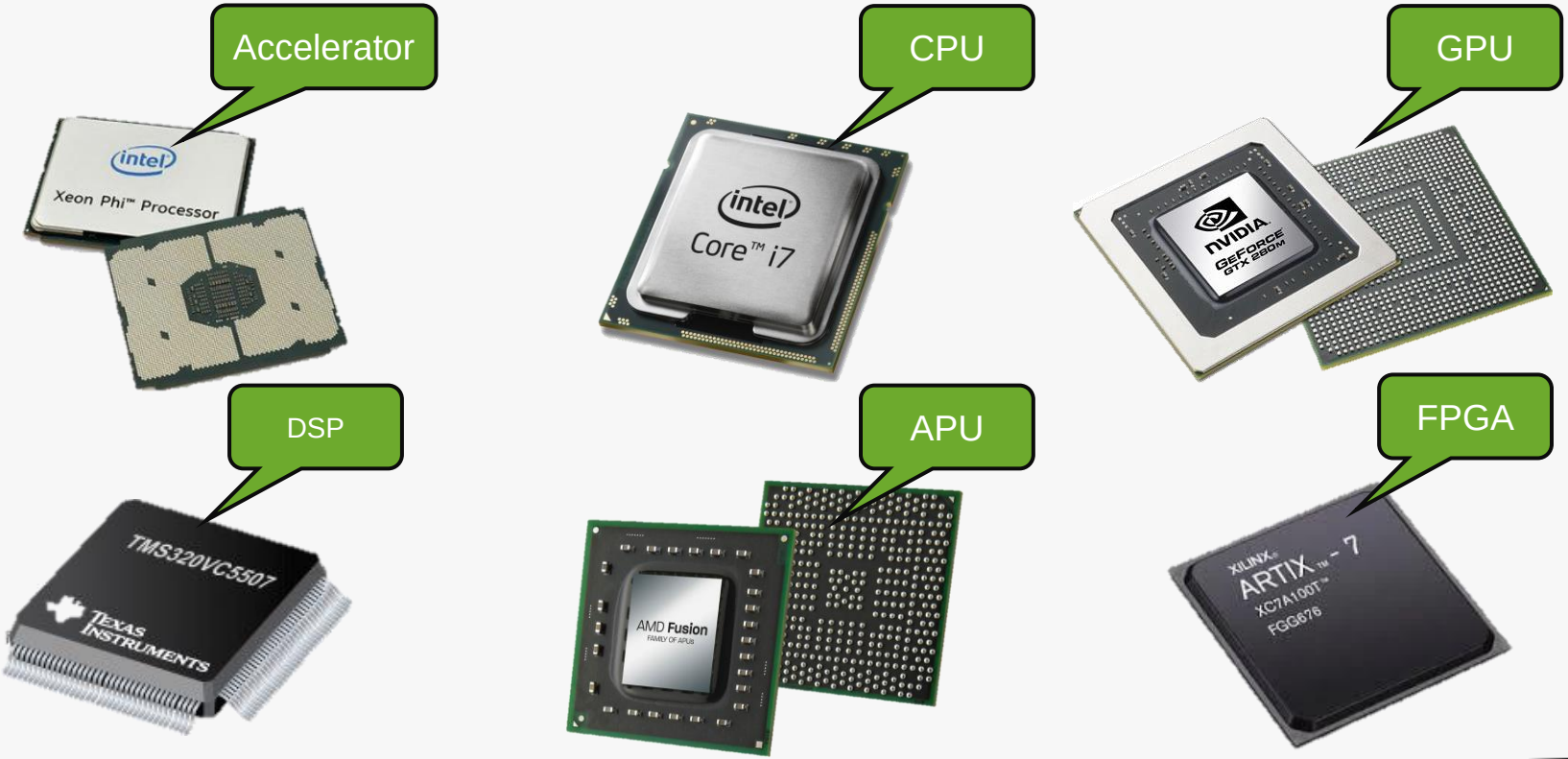
OpenCL

NVIDIA.
CUDA.

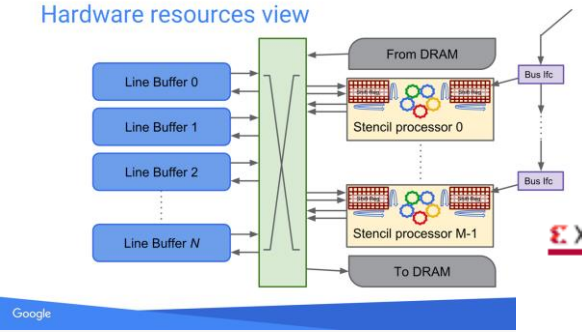
OH, East is East, and West is West,
and never the twain shall meet...

-Rudyard Kipling

Heterogeneous Devices

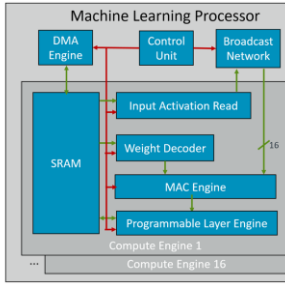


Hot Chips



Arm's ML processor: Summary

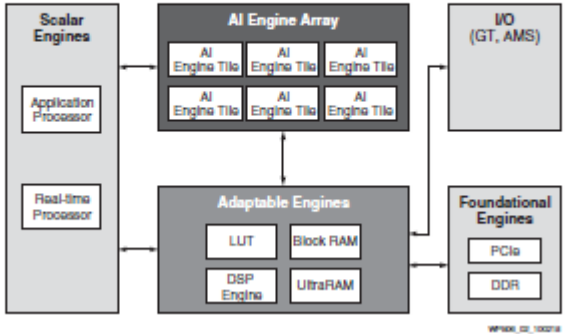
- 16 Compute Engines
- ~ 4 TOP/s of convolution throughput (at 1 GHz)
- Targeting > 3 TOP/W in 7nm and ~2.5mm²
- 8-bit quantized integer support
- 1MB of SRAM
- roid NNAPI and



XILINX.

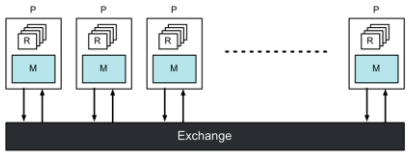
Xilinx AI Engines and Their Applications

018



Pure distributed machine with compiled communica

- Static partitioning of work and memory
- Threads hide only local latencies (arithmetic, memory, branch)
- Deterministic communication over a stateless "exchange"



ScaledML 2018

13

XAVIER INNOVATIONS

World's First Autonomous Machines Processor

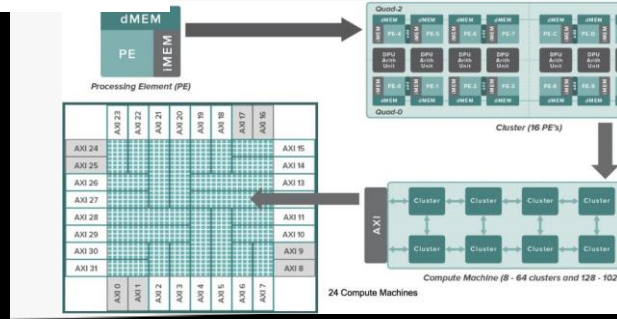
Carmel CPU	Volta GPU	DLA	PVA	ISP	MM-Accelerators
8 custom cores, ARM v8.2	512 CUDA Tensor Cores 22.6 Int8 DL TOPs	Int8/Int16/FP16 11.4 DL Int8 TOPs	7-slot VLIW 1.7 TOPs	Native HDR Processing TNR 2.4 GPIX/sec	Stereo, Optical Flow, LDC

High Speed I/Os : >40GB/s of IO Bandwidth

Designed for Safety & Resiliency : ISO26262; ASIL-C

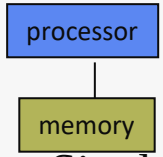
Enhanced Security

Optimized for Energy Efficiency; TSMC 12FFN

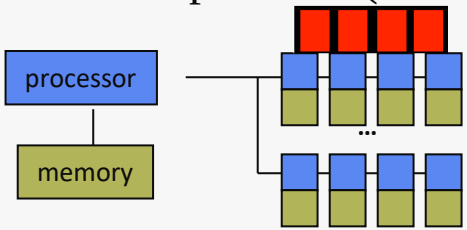


Fundamental Parallel Architecture Types

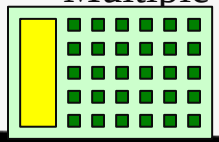
- Uniprocessor
  - Scalar processor



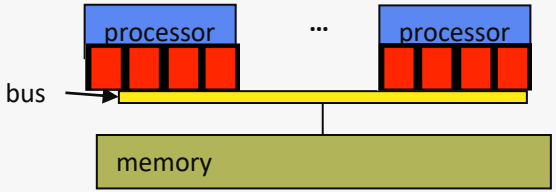
  - Single Instruction Multiple Data (SIMD)



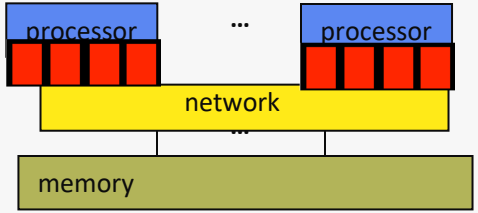
  - Single Instruction Multiple Thread (SIMT)



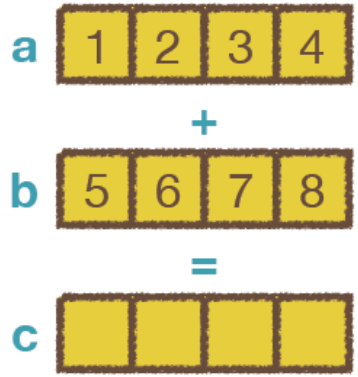
- Shared Memory Multiprocessor (SMP)
  - Shared memory address space
  - Bus-based memory system

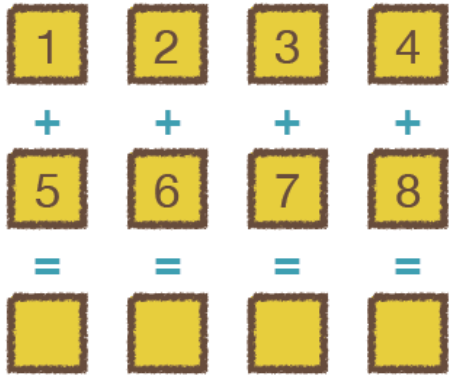


  - Interconnection network



SIMD vs SIMT



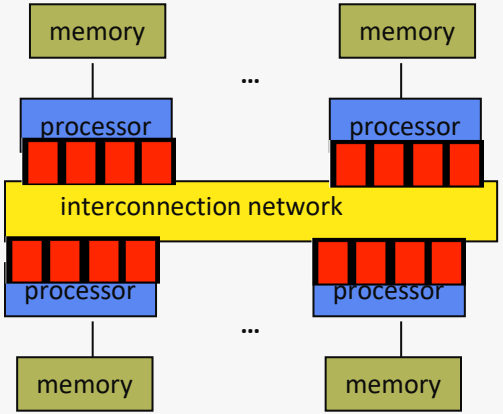



```
__m128 a = _mm_set_ps (4, 3, 2, 1);
__m128 b = _mm_set_ps (8, 7, 6, 5);
__m128 c = _mm_add_ps (a, b);
```

```
float a[4] = {1, 2, 3, 4},
      b[4] = {5, 6, 7, 8}, c[4];

// ...
// Define a compute kernel, which
// a fine-grained thread executes.
{
  int id = ... ;  // my thread ID
  c[id] = a[id] + b[id];
}
```

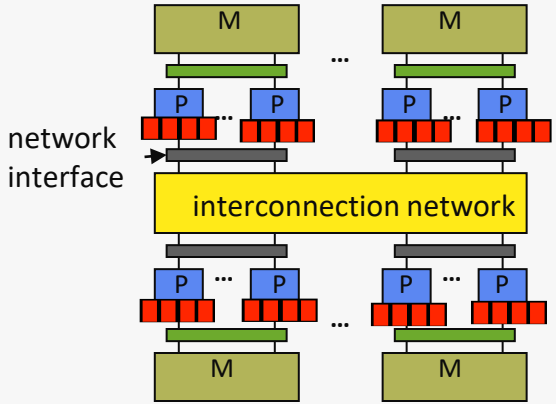
Distributed and network Parallel Architecture Types

- Distributed Memory Multiprocessor
  - Message passing between nodes



  - Massively Parallel Processor (MPP)
    - Many, many processors

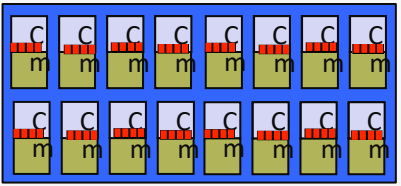
- Cluster of SMPs
  - Shared memory addressing within SMP node
  - Message passing between SMP nodes

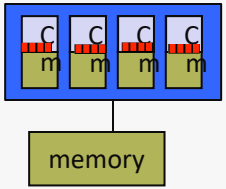


  - Can also be regarded as MPP if processor number is large

Modern Parallel Architecture

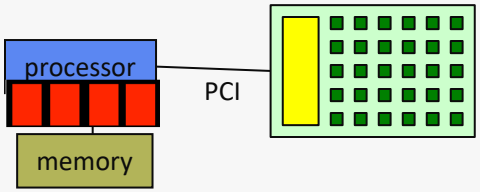
- Multicore Manycore
  - Manycore vs Multicore CPU



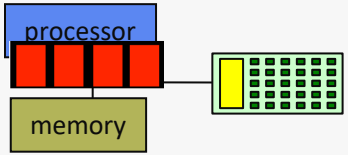


cores can be hardware multithreaded (hyperthread)

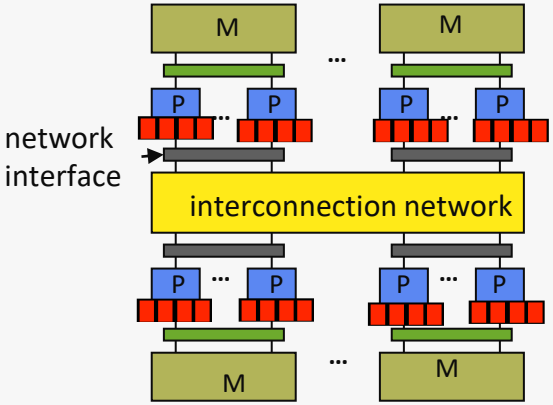
  - Heterogeneous: CPU + GPU



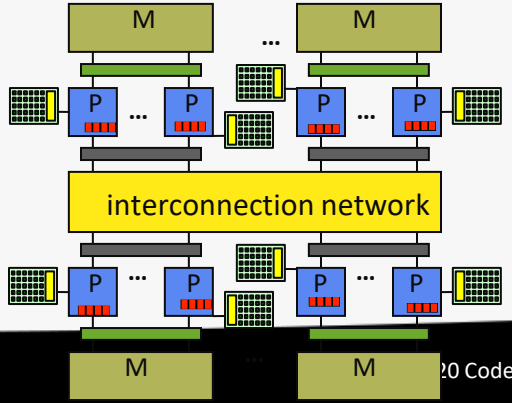
  - Heterogeneous: "Fused" CPU + GPU



- Heterogeneous: CPU+Manycore CPU

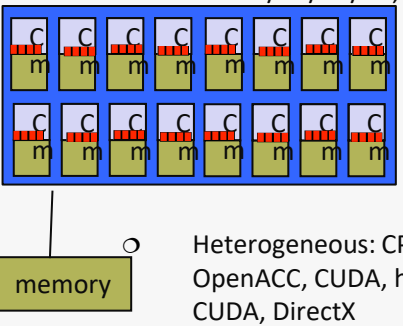


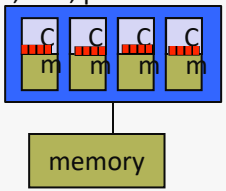
- Heterogeneous: Multicore SMP+GPU Cluster



Modern Parallel Programming model

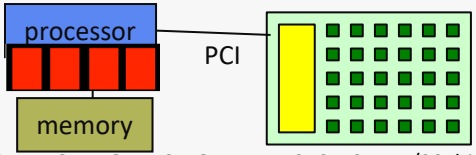
- Multicore Manycore
  - Manycore vs Multicore CPU: OpenCL, OpenMP, SYCL, C++11/14/17/20, TBB, Cilk, pthread



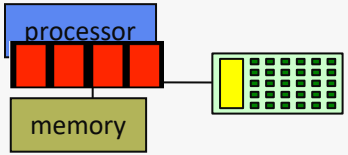


cores can be hardware multithreaded (hyperthread)

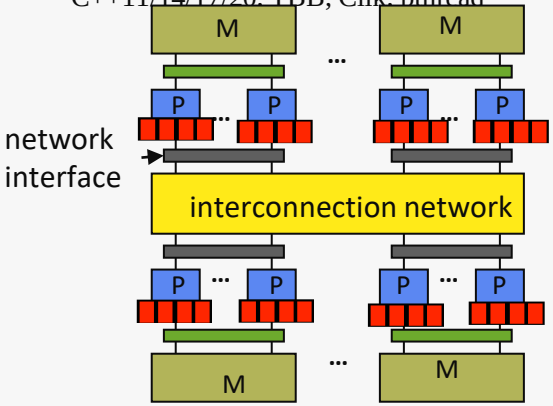
  - Heterogeneous: CPU + GPU: OpenCL, OpenMP, SYCL, C++17/20, OpenACC, CUDA, hip, RocM, C++ AMP, Intrinsics, OpenGL, Vulkan, CUDA, DirectX



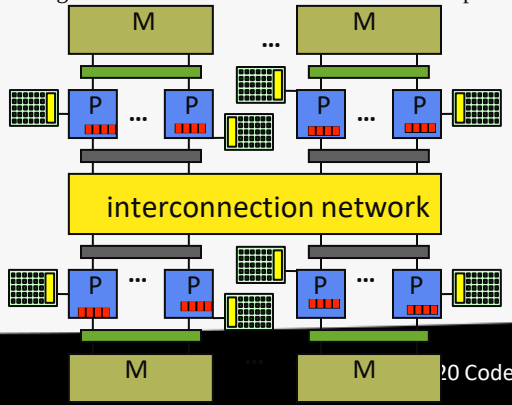
  - Heterogeneous: "Fused" CPU + GPU: OpenCL, OpenMP, SYCL, C++17/20, hip, RocM, Intrinsics, OpenGL, Vulkan, DirectX



- Heterogeneous: CPU+Manycore CPU: OpenCL, OpenMP, SYCL, C++11/14/17/20, TBB, Cilk, pthread

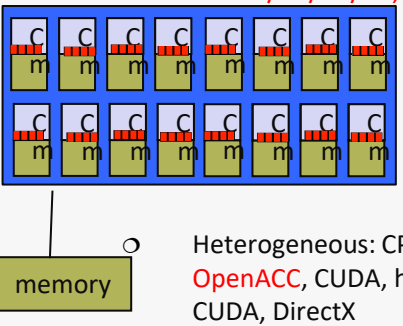


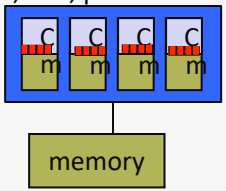
- Heterogeneous: Multicore SMP+GPU Cluster: OpenCL, OpenMP, SYCL, C++17/20



Which Programming model works on all the Architectures?Is there a pattern?

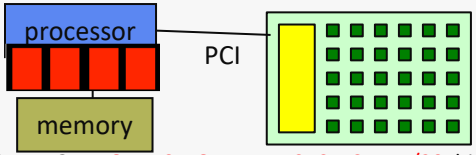
- Multicore Manycore
  - Manycore vs Multicore CPU: OpenCL, OpenMP, SYCL, C++11/14/17/20, TBB, Cilk, pthread



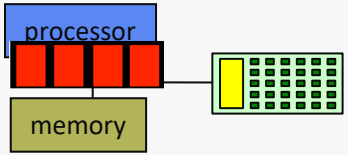


cores can be hardware multithreaded (hyperthread)

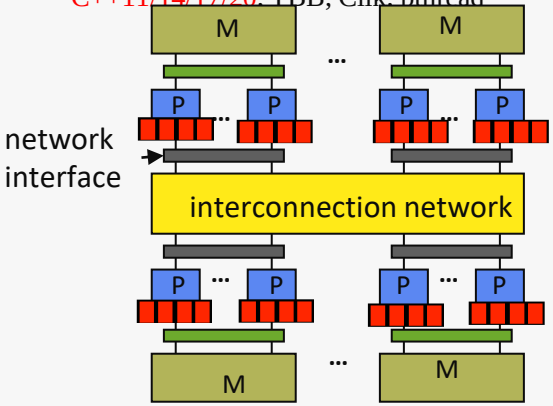
  - Heterogeneous: CPU + GPU: OpenCL, OpenMP, SYCL, C++17/20, OpenACC, CUDA, hip, RocM, C++ AMP, Intrinsics, OpenGL, Vulkan, CUDA, DirectX



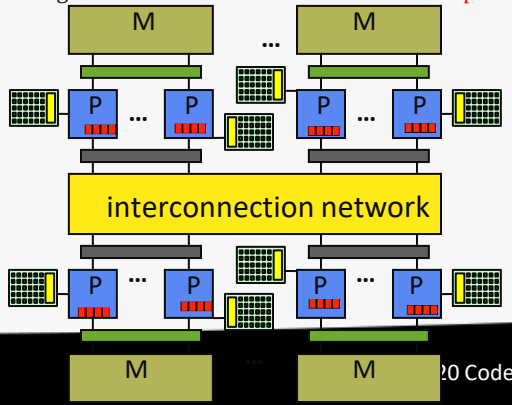
  - Heterogeneous: "Fused" CPU + GPU: OpenCL, OpenMP, SYCL, C++17/20, hip, RocM, Intrinsics, OpenGL, Vulkan, DirectX



- Heterogeneous: CPU+Manycore CPU: OpenCL, OpenMP, SYCL, C++11/14/17/20, TBB, Cilk, pthread



- Heterogeneous: Multicore SMP+GPU Cluster: OpenCL, OpenMP, SYCL, C++17/20



To support all the different parallel architectures

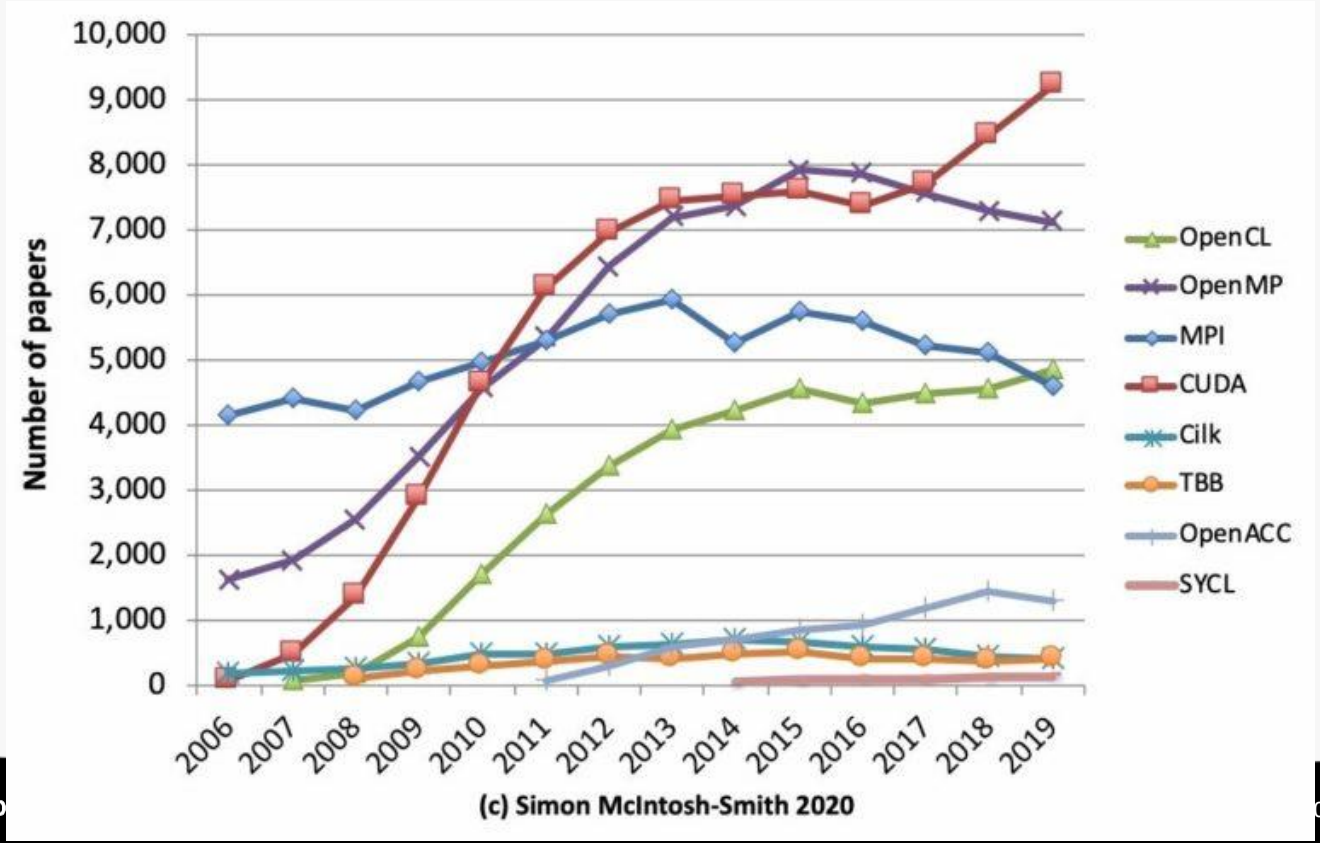
- With a single source code base
- And if you also want it to be an International Open Specification
- And if you want it to be growing with the architectures

- You really only have a few choices

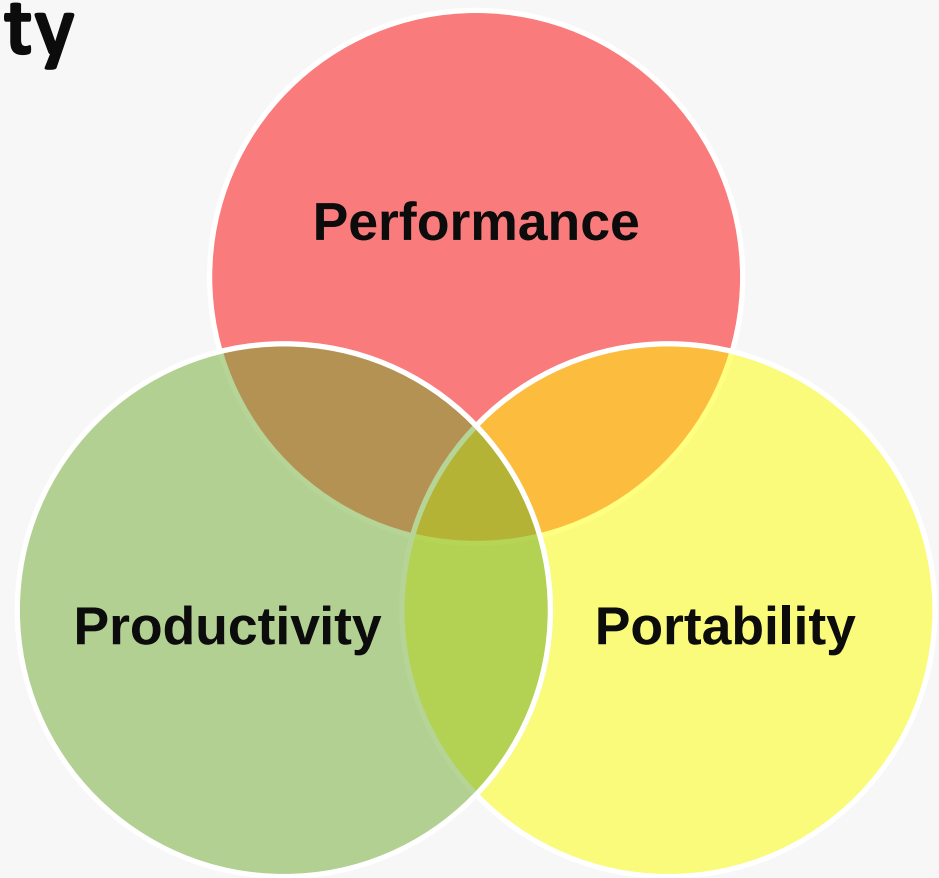
Act 2

The four horsemen of heterogeneous programming

Simon Mcintosh-Smith annual language citations



The Reality



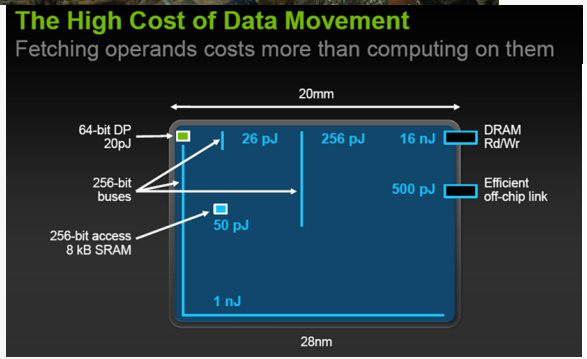
Long Answer

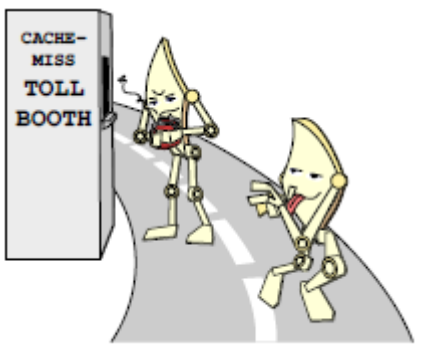
- The right programming model can allow you to express a problem in a way which adapts to different architectures

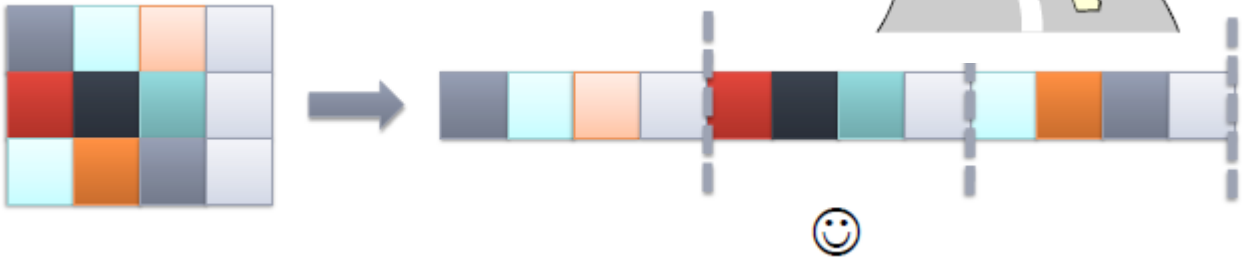
Use the right abstraction now

Abstraction	How is it supported
Cores	C++11/14/17 threads, async
HW threads	C++11/14/17 threads, async
Vectors	Parallelism TS2
Atomic, Fences, lockfree, futures, counters, transactions	C++11/14/17 atomics, Concurrency TS1, Transactional Memory TS1
Parallel Loops	Async, TBB:parallel_invoke, C++17 parallel algorithms, for_each
Heterogeneous offload, fpga	OpenCL, SYCL, HSA, OpenMP/ACC, Kokkos, Raja, CUDA
Distributed	HPX, MPI, UPC++
Caches	C++17 false sharing support
Numa	OpenMP/ACC?
TLS	?
Exception handling in concurrent environment	?

The Four Horsemen

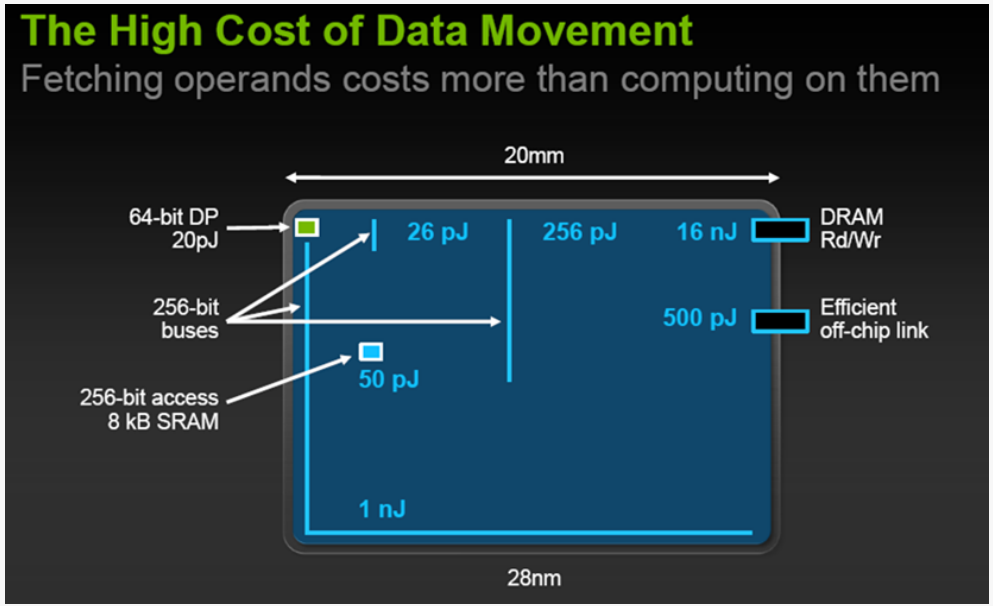






Socket 0								Socket 1							
Core 0				Core 1				Core 0				Core 1			
0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
0	4			1	5			2	6			3	7		

Cost of Data Movement



Credit: Bill Dally, Nvidia, 2010

- 64bit DP Op:
  - 20pJ
- 4x64bit register read:
  - 50pJ
- 4x64bit move 1mm:
  - 26pJ
- 4x64bit move 40mm:
  - 1nJ
- 4x64bit move DRAM:
  - 16nJ

Implicit vs Explicit Data Movement

Examples:

- SYCL, C++ AMP

Implementation:

- Data is moved to the device implicitly via cross host CPU / device data structures

Here we're using C++ AMP as an example

```
array_view<float> ptr;
extent<2> e(64, 64);
parallel_for_each(e, [=](index<2> idx)
restrict(amp) {
  ptr[idx] *= 2.0f;
});
```

Examples:

- OpenCL, CUDA, OpenMP

Implementation:

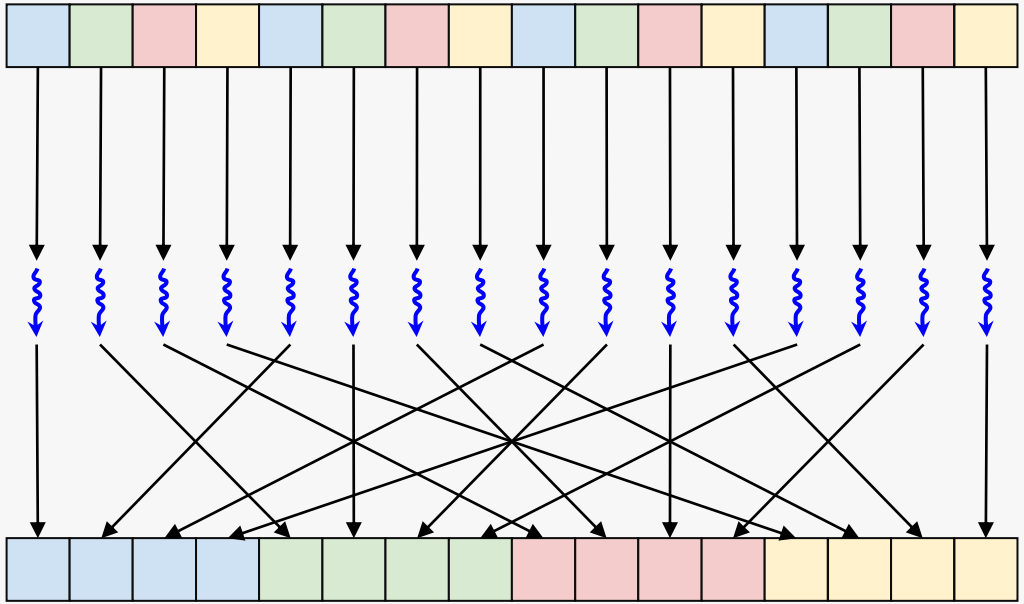
- Data is moved to the device via explicit copy APIs

Here we're using CUDA as an example

```
float *h_a = { … }, d_a;
cudaMalloc((void **)&d_a, size);
cudaMemcpy(d_a, h_a, size,
  cudaMemcpyHostToDevice);
vec_add<<<64, 64>>>(a, b, c);
cudaMemcpy(d_a, h_a, size,
  cudaMemcpyDeviceToHost);
```

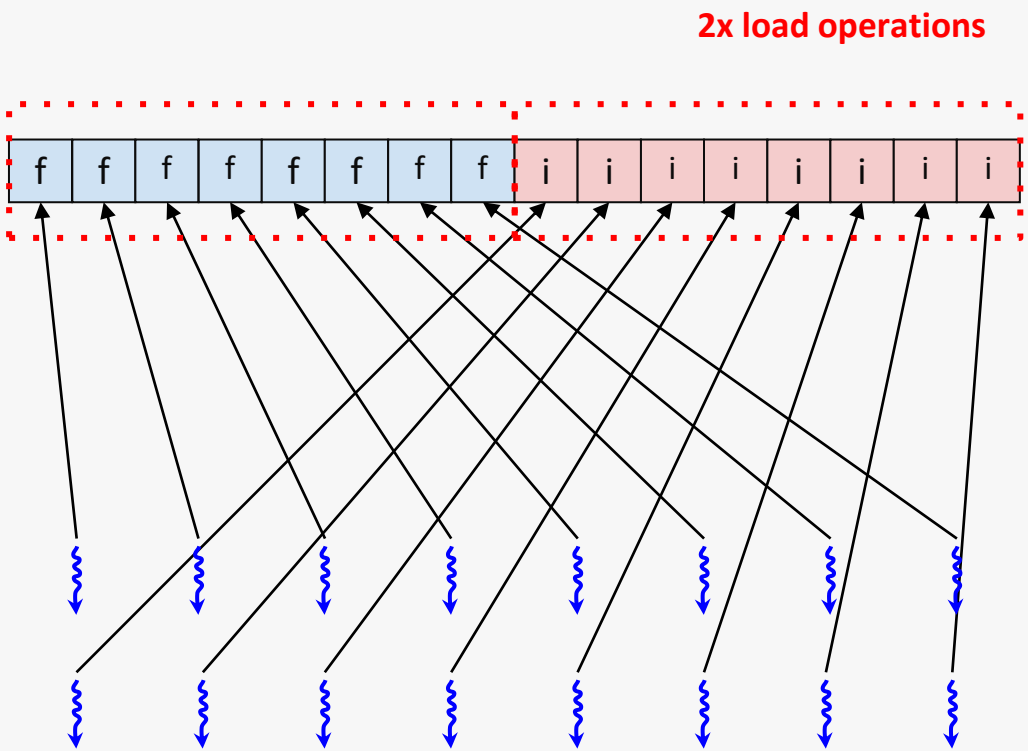
Row-major vs column-major

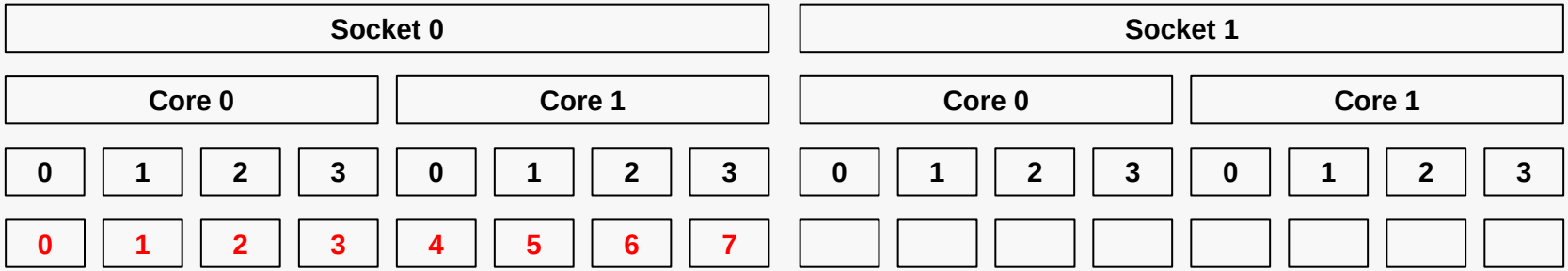
```
int x = globalId[0];
int y = globalId[1];
int stride = 4;

out[(x * stride) + y] =
  in[(y * stride) + x];
```



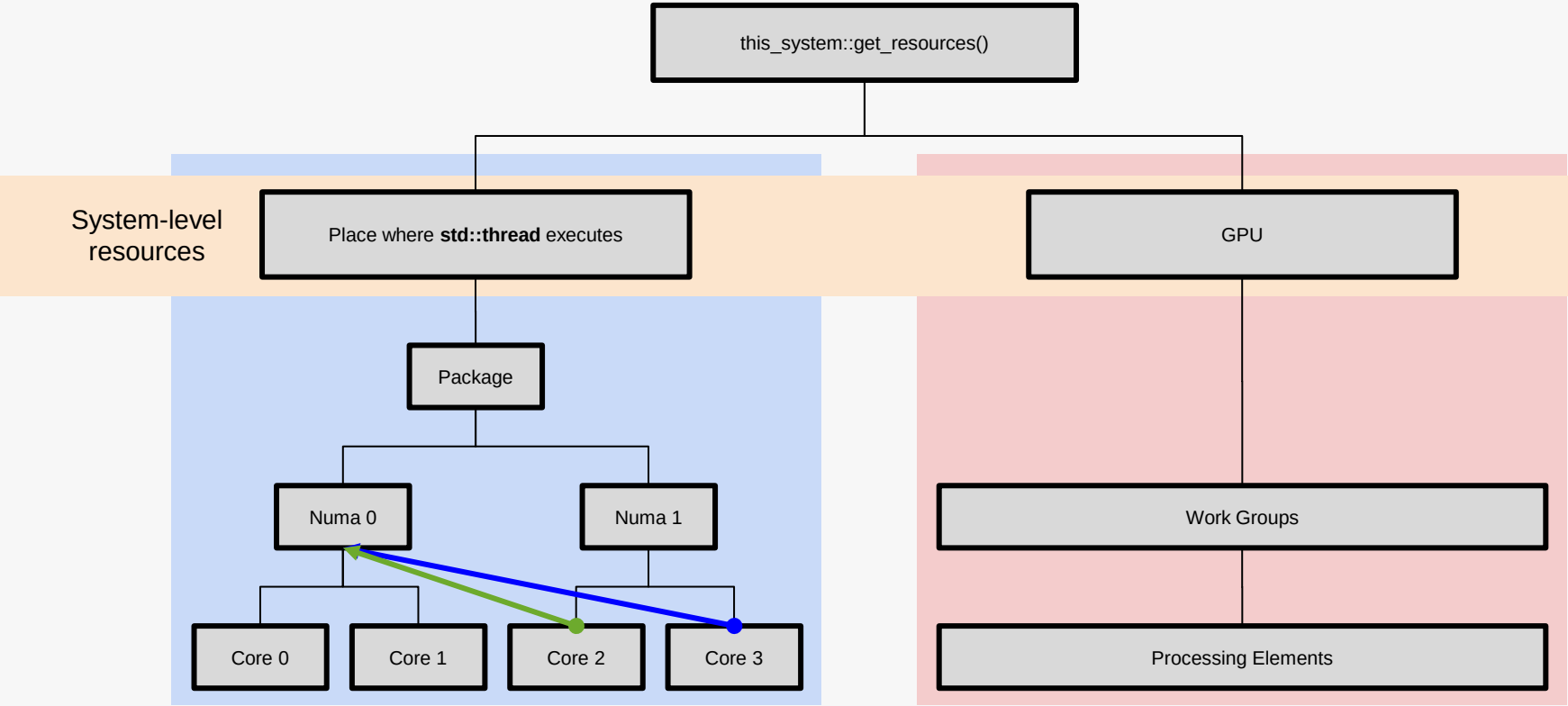
AoS vs SoA

```
struct str {
  float f[N];
  int i[N];
};

str s;

… = s.f[globalId];

… = s.i[globalId];
```






```
{
  auto exec = execution::execution_context{execRes}.executor();

  auto affExec = execution::require(exec, execution::bulk,
    execution::bulk_execution_affinity.compact);

  affExec.bulk_execute([](std::size_t i, shared s) {
    func(i);
  }, 8, sharedFactory);
}
```



relativeLatency = affinity_query<read, latency>(core2, numa0) > affinity_query<read, latency>(core3, numa0)

Act 3

C++, OpenCL, OpenMP, SYCL

Example: SAXPY

Single-precision **a** times **x p**lus **y vector addition**

$$\mathbf{y} = \alpha\mathbf{x} + \mathbf{y}$$

Serial SAXPY Implementation

```
1  void saxpy_serial(
2      size_t n,           // the number of elements in the vectors
3      float a,            // scale factor
4      const float x[],    // the first input vector
5      float y[]           // the output vector and second input vector
6  ) {
7      for (size_t i = 0; i < n; ++i)
8          y[i] = a * x[i] + y[i];
9  }
```

OpenCL

OpenMP

SYCL

CUDA

The Quiet Revolution

Kokkos

HPX

Raja

Boost.Compute

Parallel/concurrency before C++11 (C++98)

	Asynchronus Agents	Concurrent collections	Mutable shared state	Heterogeneous (GPUs, accelerators, FPGA, embedded AI processors)
summary	tasks that run independently and communicate via messages	operations on groups of things, exploit parallelism in data and algorithm structures	avoid races and synchronizing objects in shared memory	Dispatch/offload to other nodes (including distributed)
examples	GUI,background printing, disk/net access	trees, quicksorts, compilation	locked data(99%), lock-free libraries (wizards), atomics (experts)	Pipelines, reactive programming, offload,, target, dispatch
key metrics	responsiveness	throughput, many core scalability	race free, lock free	Independent forward progress,, load-shared
requirement	isolation, messages	low overhead	composability	Distributed, heterogeneous
today's abstractions	POSIX threads, win32 threads, OpenCL, vendor intrinsic	openmp, TBB, PPL, OpenCL, vendor intrinsic	locks, lock hierarchies, vendor atomic instructions, vendor intrinsic	OpenCL, CUDA

Parallel/concurrency after C++11

	Asynchronus Agents	Concurrent collections	Mutable shared state	Heterogeneous (GPUs, accelerators, FPGA, embedded AI processors)
summary	tasks that run independently and communicate via messages	operations on groups of things, exploit parallelism in data and algorithm structures	avoid races and synchronizing objects in shared memory	Dispatch/offload to other nodes (including distributed)
examples	GUI,background printing, disk/net access	trees, quicksorts, compilation	locked data(99%), lock-free libraries (wizards), atomics (experts)	Pipelines, reactive programming, offload,, target, dispatch
key metrics	responsiveness	throughput, many core scalability	race free, lock free	Independent forward progress,, load-shared
requirement	isolation, messages	low overhead	composability	Distributed, heterogeneous
today's abstractions	C++11: thread,lambda function, TLS, Async	C++11: Async, packaged tasks, promises, futures, atomics	C++11: locks, memory model, mutex, condition variable, atomics, static init/term	C++11: lambda

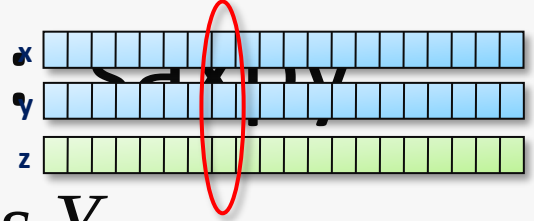
Parallel/concurrency after C++14

	Asynchronus Agents	Concurrent collections	Mutable shared state	Heterogeneous
summary	tasks that run independently and communicate via messages	operations on groups of things, exploit parallelism in data and algorithm structures	avoid races and synchronizing objects in shared memory	Dispatch/offload to other nodes (including distributed)
examples	GUI,background printing, disk/net access	trees, quicksorts, compilation	locked data(99%), lock-free libraries (wizards), atomics (experts)	Pipelines, reactive programming, offload,, target, dispatch
key metrics	responsiveness	throughput, many core scalability	race free, lock free	Independent forward progress,, load-shared
requirement	isolation, messages	low overhead	composability	Distributed, heterogeneous
today's abstractions	C++11: thread,lambda function, TLS, async C++14: generic lambda	C++11: Async, packaged tasks, promises, futures, atomics,	C++11: locks, memory model, mutex, condition variable, atomics, static init/term, C++ 14: shared_lock/shared_timed_mutex, OOTA, atomic_signal_fence,	C++11: lambda C++14: none

Parallel/concurrency after C++17

	Asynchronus Agents	Concurrent collections	Mutable shared state	Heterogeneous (GPUs, accelerators, FPGA, embedded AI processors)
summary	tasks that run independently and communicate via messages	operations on groups of things, exploit parallelism in data and algorithm structures	avoid races and synchronizing objects in shared memory	Dispatch/offload to other nodes (including distributed)
today's abstractions	C++11: thread,lambda function, TLS, async C++14: generic lambda	C++11: Async, packaged tasks, promises, futures, atomics, C++ 17: ParallelSTL, control false sharing	C++11: locks, memory model, mutex, condition variable, atomics, static init/term, C++ 14: shared_lock/shared_timed_mutex, OOTA, atomic_signal_fence, C++ 17: scoped _lock, shared_mutex, ordering of memory models, progress guarantees, TOE, execution policies	C++11: lambda C++14: generic lambda C++17: progress guarantees, TOE, execution policies

Example: Saxpy



- Saxpy == *Scalar Alpha X Plus Y*
  - *Scalar multiplication and vector addition*

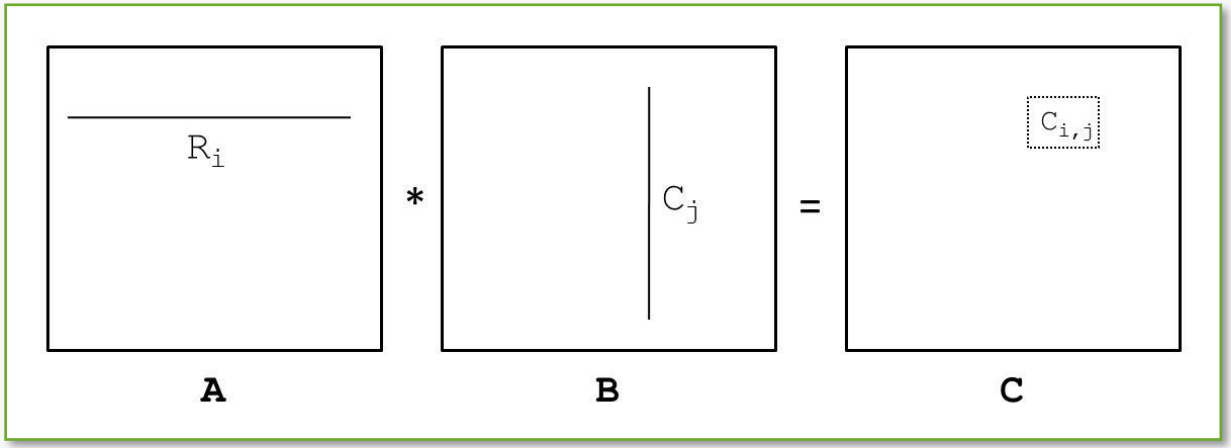
```
for (int i=0; i<n; i++)
   z[i] = a * x[i] + y[i];
```

Parallel

```
int start = …;
int end   = …;
for (int t=0; t<NumThreads; t++)
{
   thread(
     [&z,x,y,a,start,end]() -> void
     {
       for (int i = start; i < end; i++)
         z[i] = a * x[i] + y[i];
     }
   );

   start += …;
   end   += …;
}
```

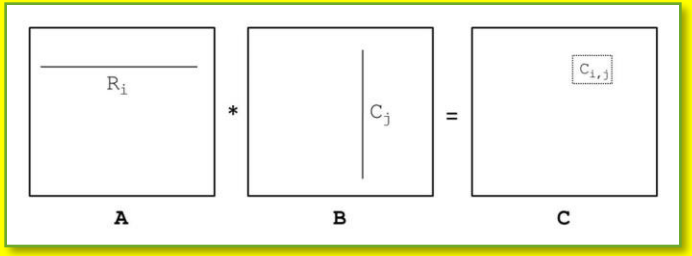
Demo #3: a complete example

- Matrix multiply…



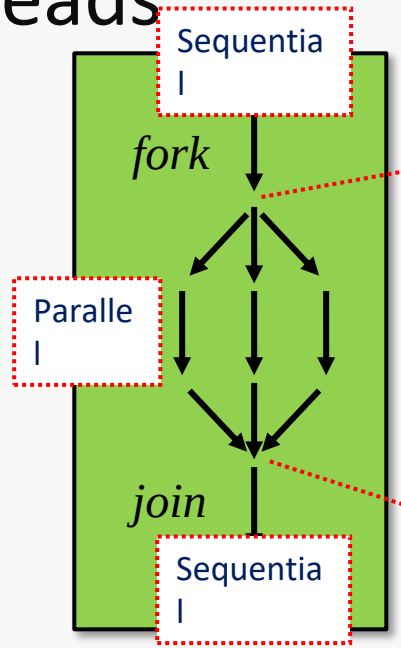
Sequential version…

```
//
// Naïve, triply-nested sequential solution:
//
for (int i = 0; i < N; i++)
{
   for (int j = 0; j < N; j++)
   {
      C[i][j] = 0.0;

      for (int k = 0; k < N; k++)
         C[i][j] += (A[i][k] * B[k][j]);
   }
}
```



Structured ("fork-join") parallelism

- A common pattern when creating multiple threads



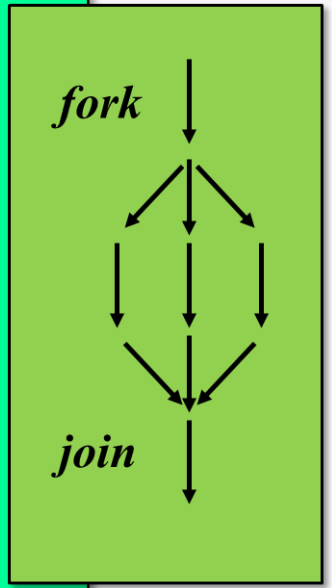

```
#include <vector>

std::vector<std::thread>  threads;

int cores = std::thread::hardware_concurrency();

for (int i=0; i<cores; ++i)  // 1 per core:
{
  auto code = []() { DoSomeWork(); };
  threads.push_back( thread(code) );
}
```

```
for (std::thread& t : threads) // new range-based for:
  t.join();
```

Parallel solution

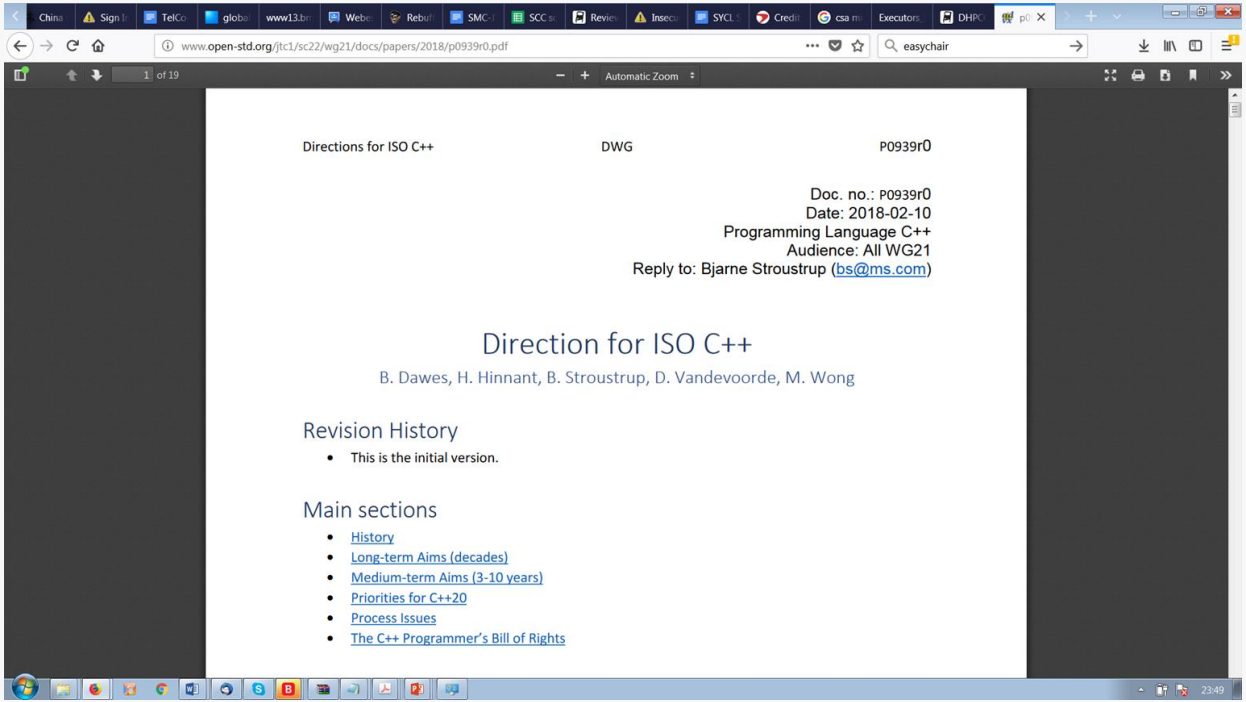
```
int rows  = N / numthreads;
int extra = N % numthreads;
int start = 0;      // each thread does [start..end)
int end   = rows;

vector<thread>  workers;

for (int t = 1; t <= numthreads; t++)
{
  if (t == numthreads)  // last thread does extra rows:
    end += extra;

  workers.push_back( thread([start, end, N, &C, &A, &B]()
  {
    for (int i = start; i < end; i++)
      for (int j = 0; j < N; j++)
      {
        C[i][j] = 0.0;
        for (int k = 0; k < N; k++)
          C[i][j] += (A[i][k] * B[k][j]);
      }
  }));

  start = end;
  end   = start + rows;
}
```

```
// 1 thread per core:
numthreads = thread::hardware_concurrency();
```




```
for (thread& t : workers)
  t.join();
```

C++ Directions Group: P2000



Directions for ISO C++ DWG P0939r0

Doc. no.: P0939r0
Date: 2018-02-10
Programming Language C++
Audience: All WG21
Reply to: Bjarne Stroustrup (bs@ms.com)

Direction for ISO C++

B. Dawes, H. Hinnant, B. Stroustrup, D. Vandevoorde, M. Wong

Revision History

- This is the initial version.

Main sections

- History
- Long-term Aims (decades)
- Medium-term Aims (3-10 years)
- Priorities for C++20
- Process Issues
- The C++ Programmer's Bill of Rights

P2000:Modern hardware

We need better support for modern hardware, such as executors/execution context, affinity support in C++ leading to heterogeneous/distributed computing support, ...

Parallel/concurrency aiming for C++20

	Asynchronus Agents	Concurrent collections	Mutable shared state	Heterogeneous/Distributed
today's abstractions	C++11: thread,lambda function, TLS, async C++ 20: Jthreads +interrupt _token, coroutines	C++11: Async, packaged tasks, promises, futures, atomics, C++ 17: ParallelSTL, control false sharing C++ 20: Vec execution policy, Algorithm un-sequenced policy, span	C++11: locks, memory model, mutex, condition variable, atomics, static init/term, C++ 14: shared_lock/shared_timed_mutex, OOTA, atomic_signal_fence, C++ 17: scoped _lock, shared_mutex, ordering of memory models, progress guarantees, TOE, execution policies C++20: atomic_ref, Latches and barriers, atomic<shared_ptr> Atomics & padding bits Simplified atomic init Atomic C/C++ compatibility Semaphores and waiting Fixed gaps in memory model , Improved atomic flags, Repair memory model	C++11: lambda C++14: generic lambda C++17: , progress guarantees, TOE, execution policies C++20: atomic_ref,, span

Parallel/Concurrency beyond C++20: C++23

	Asynchronus Agents	Concurrent collections	Mutable shared state	Heterogeneous/DIstributed
today's abstractions	C++11: thread,lambda function, TLS, async C++14: generic lambda C++ 20: Jthreads +interrupt _token C++23: networking, asynchronous algorithm, reactive programming, EALS, async2, executors	C++11: Async, packaged tasks, promises, futures, atomics, C++ 17: ParallelSTL, control false sharing C++ 20: Vec execution policy, Algorithm un-sequenced policy span C++23: SMD<T>,new futures, concurrent vector,task blocks, unordered associative containers, two-way executors with lazy sender-receiver models, concurrent exception handling, executors, mdspan	C++11: … C++ 14: … C++ 17: … C++20: atomic_ref, Latches and barriers atomic<shared_ptr> Atomics & padding bits Simplified atomic init Atomic C/C++ compatibility Semaphores and waiting Fixed gaps in memory model , Improved atomic flags , Repair memory model C++23: hazard_pointers, rcu/snapshot, concurrent queues, counters, upgrade lock, TM lite, more lock-free data structures, asymmetric fences	C++17: , progress guarantees, TOE, execution policies C++20: atomic_ref, mdspan, C++23: SIMD<T>,affinity, pipelines, EALS, freestanding/embedded support well specified, mapreduce, ML/AI, reactive programming executors, mdspan

C++23: continue C++20

- Library support for coroutines
- Further Conceptifying Standard Library
- Further Range improvements (e.g., application of ranges to parallel algorithms and operations on containers and integration with coroutines)
- A modular standard library

After C++20

- Much more libraries
  - Audio
  - Linear Algebra
  - Graph data structures
  - Tree Data structures
  - Task Graphs
  - Differentiation
  - Reflection
  - Light-weight transactional locks
  - A new future and/or a new async
  - Statistics Library
  - Array style programming through mdspan
- Machine learning support
- Executors
- Networking
- Pattern Matching
- Better support for C++Tooling ecosystem
- Further support for heterogeneous programming
- Graphics
- Better definition of freestanding
- Education dependency curriculum

After C++23

- Reflection
- Pattern matching
- C++ ecosystem
- What about Contracts?

What have we achieved so far for C++20?

	Depends on	Current target (estimated, could slip)
Concepts		C++20 (adopted, including convenience syntax)
Contracts		C++20 (a[illegible])
Ranges		C++20 (adopted)
Coroutines		C++20
Modules		C++20
Reflection		TS in C++20 timeframe, IS in C++23
Executors		Lite in C++20 timeframe, Full in C++23
Networking	Executors, and possibly Coroutines	C++23
future.then, async2	Executors	

SYCL Ecosystem

- ComputeCpp - https://codeplay.com/products/computesuite/computecpp
- triSYCL - https://github.com/triSYCL/triSYCL
- SYCL - http://sycl.tech
- SYCL ParallelSTL - https://github.com/KhronosGroup/SyclParallelSTL
- VisionCpp - https://github.com/codeplaysoftware/visioncpp
- SYCL-BLAS - https://github.com/codeplaysoftware/sycl-blas
- TensorFlow-SYCL - https://github.com/codeplaysoftware/tensorflow
- Eigen http://eigen.tuxfamily.org

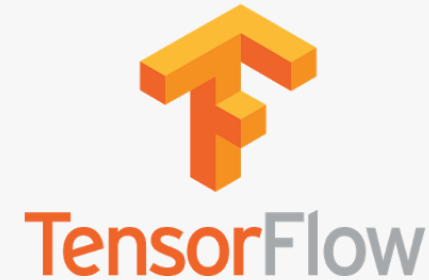
Eigen Linear Algebra Library

SYCL backend in mainline
Focused on Tensor support, providing support for machine learning/CNNs
Equivalent coverage to CUDA
Working on optimization for various hardware architectures (CPU, desktop and mobile GPUs)
https://bitbucket.org/eigen/eigen/

TensorFlow

- SYCL backend support for all major CNN operations
- Complete coverage for major image recognition networks
  - GoogLeNet, Inception-v2, Inception-v3, ResNet, ….
- Ongoing work to reach 100% operator coverage and optimization for various hardware architectures (CPU, desktop and mobile GPUs)

https://github.com/tensorflow/tensorflow



SYCL Ecosystem

- Single-source heterogeneous programming using STANDARD C++
  - Use C++ templates and lambda functions for host & device code
  - Layered over OpenCL
- Fast and powerful path for bring C++ apps and libraries to OpenCL
  - C++ Kernel Fusion - better performance on complex software than hand-coding
  - Halide, Eigen, Boost.Compute, SYCLBLAS, SYCL Eigen, SYCL TensorFlow, SYCL GTX
  - Clang, triSYCL, ComputeCpp, VisionCpp, ComputeCpp SDK …
- More information at http://sycl.tech

Developer Choice

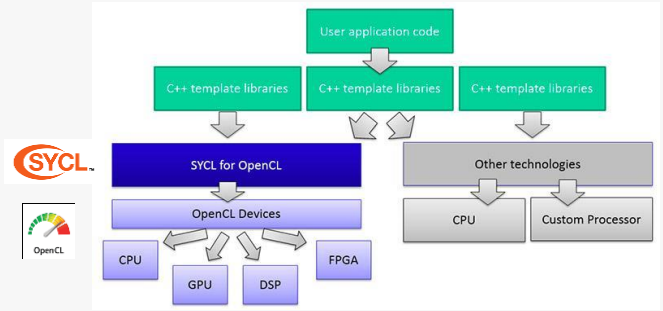
The development of the two specifications are aligned so code can be easily shared between the two approaches

C++ Kernel Language
Low Level Control
'GPGPU'-style separation of device-side kernel source code and host code

OpenCL

Single-source C++
Programmer Familiarity
Approach also taken by C++ AMP and OpenMP

SYCL™



Codeplay

Standards bodies

- HSA Foundation: Chair of software group, spec editor of runtime and debugging
- Khronos: chair & spec editor of SYCL. Contributors to OpenCL, Safety Critical, Vulkan
- ISO C++: Chair of Low Latency, Embedded WG; Editor of SG1 Concurrency TS
- EEMBC: members

Research

- Members of EU research consortiums: PEPPHER, LPGPU, LPGPU2, CARP
- Sponsorship of PhDs and EngDs for heterogeneous programming: HSA, FPGAs, ray-tracing
- Collaborations with academics
- Members of HiPEAC

Open source

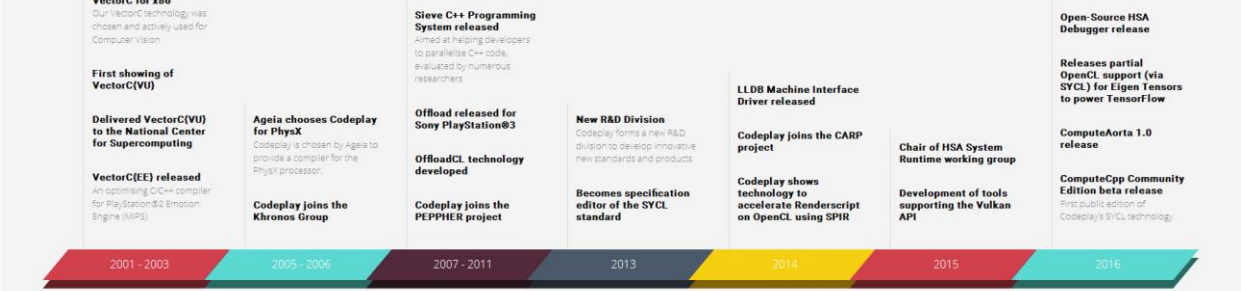
- HSA LLDB Debugger
- SPIR-V tools
- RenderScript debugger in AOSP
- LLDB for Qualcomm Hexagon
- TensorFlow for OpenCL
- C++ 17 Parallel STL for SYCL
- VisionCpp: C++ performance-portable programming model for vision

Presentations

- Building an LLVM back-end
- Creating an SPMD Vectorizer for OpenCL with LLVM
- Challenges of Mixed-Width Vector Code Gen & Scheduling in LLVM
- C++ on Accelerators: Supporting Single-Source SYCL and HSA
- LLDB Tutorial: Adding debugger support for your target

Company

- Based in Edinburgh, Scotland
- 57 staff, mostly engineering
- License and customize technologies for semiconductor companies
- ComputeAorta and ComputeCpp: implementations of OpenCL, Vulkan and SYCL
- 15+ years of experience in heterogeneous systems tools



Codeplay build the software platforms that deliver massive performance

What our ComputeCpp users say about us

Benoit Steiner – Google TensorFlow engineer

"We at Google have been working closely with Luke and his Codeplay colleagues on this project for almost 12 months now. Codeplay's contribution to this effort has been tremendous, so we felt that we should let them take the lead when it comes down to communicating updates related to OpenCL. … we are planning to merge the work that has been done so far… we want to put together a comprehensive test infrastructure"

ONERA

"We work with royalty-free SYCL because it is hardware vendor agnostic, single-source C++ programming model without platform specific keywords. This will allow us to easily work with any heterogeneous processor solutions using OpenCL to develop our complex algorithms and ensure future compatibility"

Hartmut Kaiser - HPX

"My team and I are working with Codeplay's ComputeCpp for almost a year now and they have resolved every issue in a timely manner, while demonstrating that this technology can work with the most complex C++ template code. I am happy to say that the combination of Codeplay's SYCL implementation with our HPX runtime system has turned out to be a very capable basis for Building a Heterogeneous Computing Model for the C++ Standard using high-level abstractions."

WIGNER Research Centre for Physics

It was a great pleasure this week for us, that Codeplay released the ComputeCpp project for the wider audience. We've been waiting for this moment and keeping our colleagues and students in constant rally and excitement. We'd like to build on this opportunity to increase the awareness of this technology by providing sample codes and talks to potential users. We're going to give a lecture series on modern scientific programming providing field specific examples."

Further information

- OpenCL https://www.khronos.org/opencl/
- OpenVX https://www.khronos.org/openvx/
- HSA http://www.hsafoundation.com/
- SYCL http://sycl.tech
- OpenCV http://opencv.org/
- Halide http://halide-lang.org/
- VisionCpp https://github.com/codeplaysoftware/visioncpp

Community Edition

Available now for free!

Visit:

computecpp.codeplay.com

- Open source SYCL projects:
  - ComputeCpp SDK - Collection of sample code and integration tools
  - SYCL ParallelSTL – SYCL based implementation of the parallel algorithms
  - VisionCpp – Compile-time embedded DSL for image processing
  - Eigen C++ Template Library – Compile-time library for machine learning

All of this and more at: http://sycl.tech

We're Hiring!
codeplay.com/careers/

codeplay
THE HETEROGENEOUS SYSTEMS EXPERTS

Thanks

@codeplaysoft

info@codeplay.com

codeplay.com