



University of Stuttgart
Germany

IPVS

Institute for Parallel and
Distributed Systems

Scientific Computing



Marcel.Breyer@ipvs.uni-stuttgart.de

Marcel
Breyer

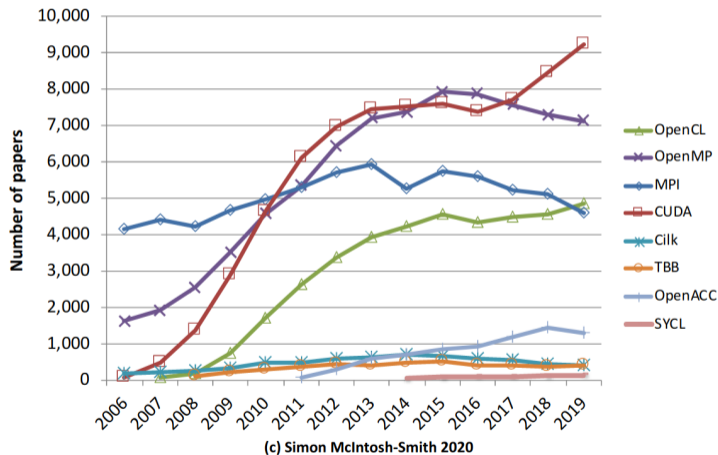
A Comparison of SYCL, OpenCL, CUDA, and OpenMP for Massively Parallel Support Vector Machine Classification on Multi-Vendor Hardware

*10th IWOCCL & SYCLcon
Mai 10-12, 2022*



Motivation - Parallel Programming Languages

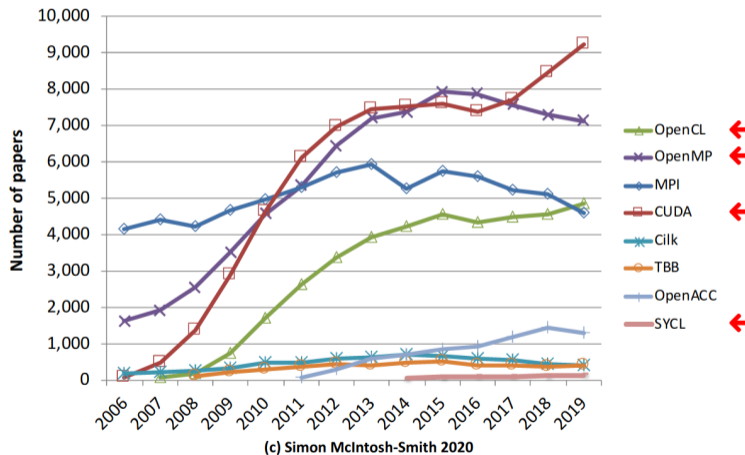
Papers mentioning parallel programming languages.
Data according to Google Scholar (April 27th 2020)



Source: <https://www.iwocl.org/wp-content/uploads/iwocl-syclcon-2020-panel-slides.pdf> (slide 2)

Motivation - Parallel Programming Languages

Papers mentioning parallel programming languages.
Data according to Google Scholar (April 27th 2020)



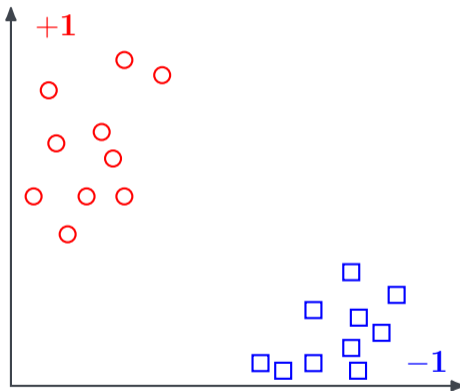
Source: <https://www.iwocl.org/wp-content/uploads/iwocl-syclcon-2020-panel-slides.pdf> (slide 2)

Example Application

1

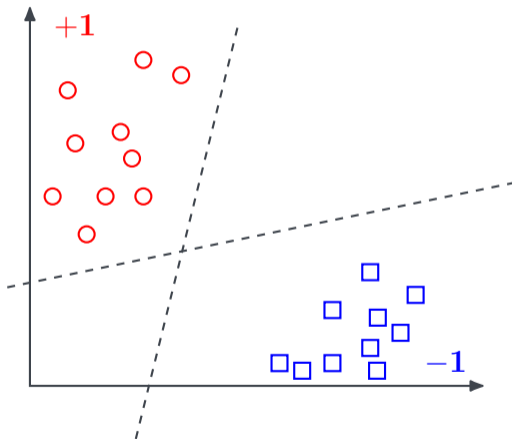
Support Vector Machine (SVM) (proposed by Boser, Guyon, and Vapnik in 1992)

supervised machine learning: binary classification



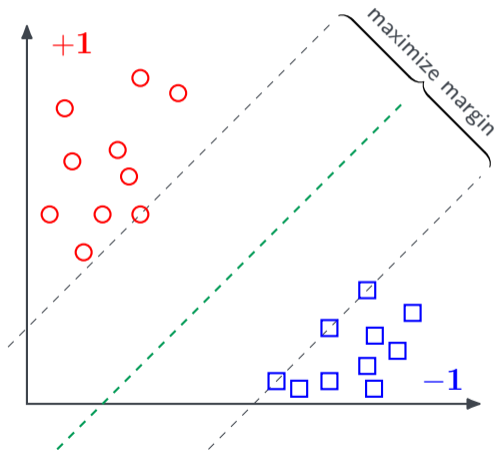
Support Vector Machine (SVM) (proposed by Boser, Guyon, and Vapnik in 1992)

supervised machine learning: binary classification



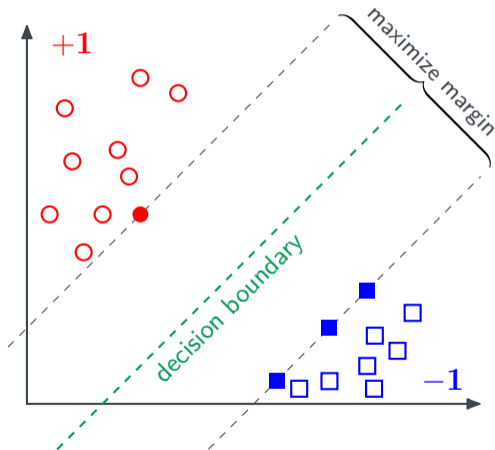
Support Vector Machine (SVM) (proposed by Boser, Guyon, and Vapnik in 1992)

supervised machine learning: binary classification



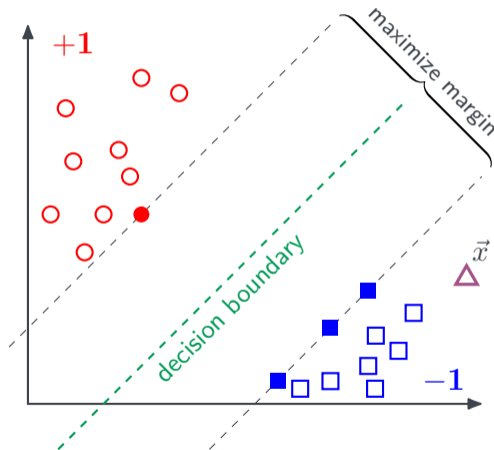
Support Vector Machine (SVM) (proposed by Boser, Guyon, and Vapnik in 1992)

supervised machine learning: binary classification



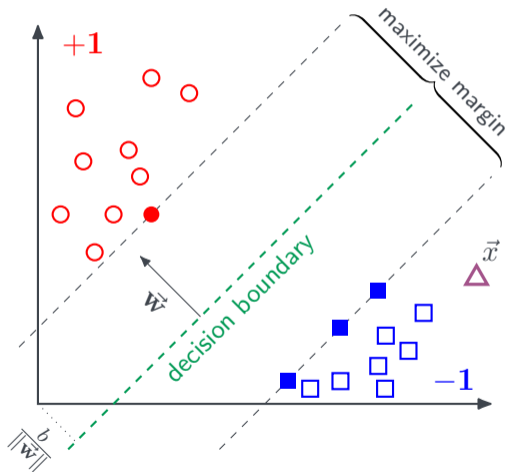
Support Vector Machine (SVM) (proposed by Boser, Guyon, and Vapnik in 1992)

supervised machine learning: binary classification



Support Vector Machine (SVM) (proposed by Boser, Guyon, and Vapnik in 1992)

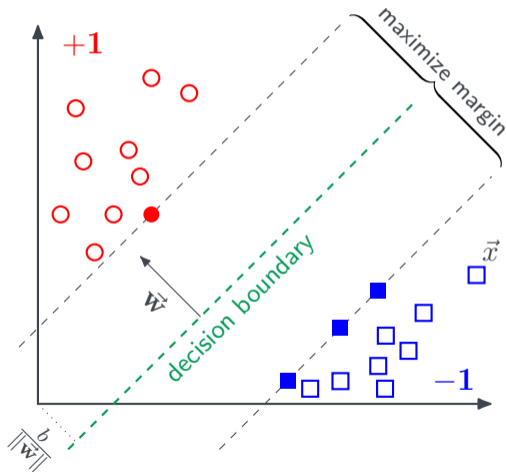
supervised machine learning: binary classification



$$y = \text{sgn}(\langle \vec{w}, \vec{x} \rangle + b)$$

Support Vector Machine (SVM) (proposed by Boser, Guyon, and Vapnik in 1992)

supervised machine learning: binary classification



$$y = \text{sgn}(\langle \vec{w}, \vec{x} \rangle + b)$$

Support Vector Machine (SVM) - Problems

- SVMs have to solve a convex quadratic problem
 - ➔ state-of-the-art: Sequential Minimal Optimization (SMO) (proposed by Platt in 1998)
 - ➔ inherently sequential algorithm

Support Vector Machine (SVM) - Problems

- SVMs have to solve a convex quadratic problem
 - ➔ state-of-the-art: Sequential Minimal Optimization (SMO) (proposed by Platt in 1998)
 - ➔ inherently sequential algorithm
- many SVM implementations modify SMO to exploit some parallelism
 - ➔ still not well suited for modern, highly parallel hardware

Support Vector Machine (SVM) - Problems

- SVMs have to solve a convex quadratic problem
 - ➔ state-of-the-art: Sequential Minimal Optimization (SMO) (proposed by Platt in 1998)
 - ➔ inherently sequential algorithm
- many SVM implementations modify SMO to exploit some parallelism
 - ➔ still not well suited for modern, highly parallel hardware

➔ *Least Squares Support Vector Machine (LS-SVM)*
(proposed by Suykens and Vandewalle in 1999)

Support Vector Machine (SVM) - Problems

- SVMs have to solve a convex quadratic problem
 - ➔ state-of-the-art: Sequential Minimal Optimization (SMO) (proposed by Platt in 1998)
 - ➔ inherently sequential algorithm
- many SVM implementations modify SMO to exploit some parallelism
 - ➔ still not well suited for modern, highly parallel hardware

➔ *Least Squares Support Vector Machine (LS-SVM)*

(proposed by Suykens and Vandewalle in 1999)

- reformulation of standard SVM to solving a system of linear equations
- massively parallel algorithms known, e.g., Conjugate Gradient (CG)

**Implemen-
tation**

2

Parallel Least Squares Support Vector Machine (PLSSVM)

- modern C++17
- single and double precision via template parameter
- backend and target platform selectable at runtime
- parallelizes matrix-vector multiplication in CG algorithm
- multi-GPU support for the linear kernel function
- drop-in replacement for LIBSVM's `svm-train` and `svm-predict` executables
- currently only binary classification and dense calculations



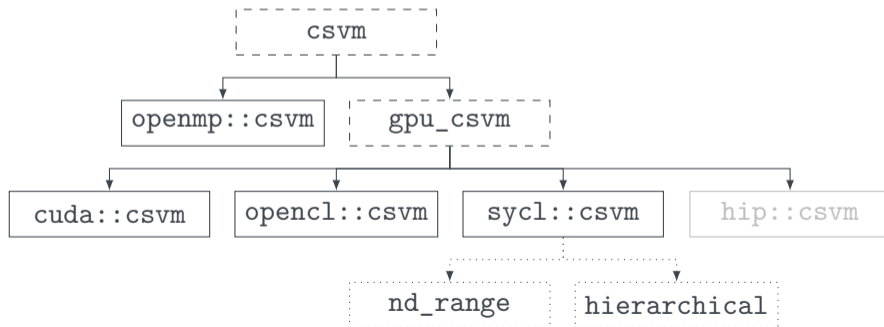
Backends Implemented in PLSSVM

OpenMP



OpenCL

SYCL



Backends Implemented in PLSSVM

OpenMP



OpenCL

SYCL

- CPU only (no target offloading for GPUs)
- only directive based constructs
- not yet optimized to the same level as the GPU backends

Backends Implemented in PLSSVM

OpenMP



OpenCL

SYCL

- optimizations: blocking, caching, padding
- block-level caching (global ↔ shared memory)
- thread-level caching (shared memory ↔ register)
- blocking sizes changeable during compilation
- Ahead-of-Time (AOT) instead of Just-in-Time (JIT) compilation

Backends Implemented in PLSSVM



- same optimizations as in CUDA
- C++ (RAII) wrapper around OpenCL handles
- custom floating point atomic functions via `atomic_cmpxchg` and `atom_cmpxchg`
- no AOT compilation, JIT only
- custom OpenCL kernel binary caching implementation

Backends Implemented in PLSSVM

The logo for OpenMP, featuring the text "OpenMP" in a light blue, sans-serif font. The "Open" is in a lighter shade of blue, and "MP" is in a slightly darker shade. A horizontal line is positioned below the "Open" part of the text.

- same optimizations as in CUDA
- DPC++ and hipSYCL supported
- other SYCL implementations (e.g., ComputeCpp, triSYCL, neoSYCL) not investigated (missing features)
- DPC++ with AOT compilation



Backends Implemented in PLSSVM

The logo for OpenMP, featuring the text "OpenMP" in a light blue, sans-serif font. The "Open" is in a lighter shade of blue, and "MP" is in a slightly darker shade. A horizontal line is positioned below the "Open" part.The OpenCL logo, featuring a semi-circular arc composed of several colored segments (green, yellow, orange, red) above the text "OpenCL" in a grey, sans-serif font. A small "TM" trademark symbol is to the right.The SYCL logo, featuring the word "SYCL" in a bold, orange, sans-serif font, enclosed within a large, orange, stylized "C" shape. A small "TM" trademark symbol is to the right.

- same optimizations as in CUDA
- DPC++ and hipSYCL supported
- other SYCL implementations (e.g., ComputeCpp, triSYCL, neoSYCL) not investigated (missing features)
- DPC++ with AOT compilation
- two implementations of the same kernel:
 - `nd_range`: directly comparable to CUDA and OpenCL
 - hierarchical: acceptable performance using hipSYCL on CPUs

The oneAPI logo, featuring a stylized, purple, 3D-like structure of spheres arranged in a vertical column, with the word "oneAPI" in a black, sans-serif font below it.The hipSYCL logo, featuring the word "hipSYCL" in a red, sans-serif font, with a red swoosh underline beneath the "hip" part.

Results

3

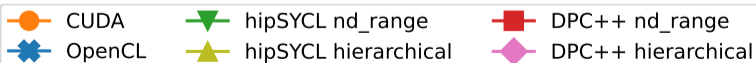
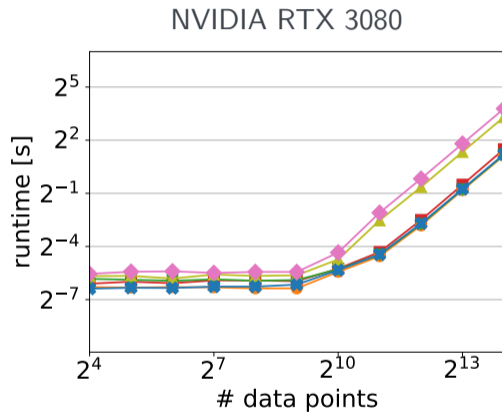
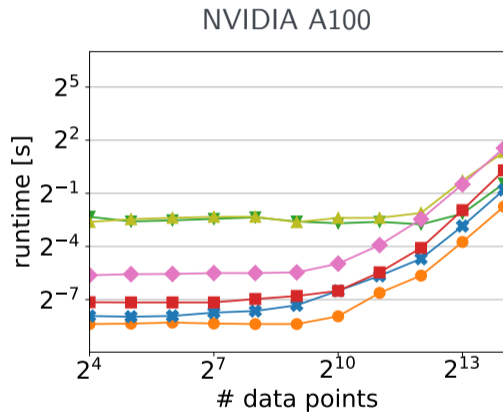
Setup - Hardware

			memory	bandwidth	FP64
GPUs	NVIDIA	A100	40 GB HBM2e	1555 GB/s	9.7 TFLOPS
		P100	16 GB HBM2	732 GB/s	4.7 TFLOPS
		RTX 3080	10 GB GDDR6X	760.3 GB/s	465.1 GFLOPS
		GTX 1080 Ti	11 GB GDDR5X	484.4 GB/s	354.4 GFLOPS
	AMD Radeon Pro VII		16 GB HBM2	1024 GB/s	6.5 TFLOPS
	Intel	UHD P630	53.8 GB DDR4	41.6 GB/s	96 GFLOPS
		Iris Xe MAX	4 GB LPDDR4x	68 GB/s	emulated
			base/boost freq.	bandwidth	# cores/# HT
CPUs	AMD	EPYC 7742	2.25/3.4 GHz	204.8 GB/s	2 · 64/2 · 128
		Ryzen TR 3960X	3.8/4.5 GHz	102.4 GB/s	24/48
	Intel	Xeon Phi 7210	1.3/1.5 GHz	102 GB/s	64/256
		Xeon E-2176G	3.7/4.7 GHz	41.6 GB/s	6/12
		Core i9-10920X	3.5/4.6 GHz	94 GB/s	12/24

Setup - Hardware

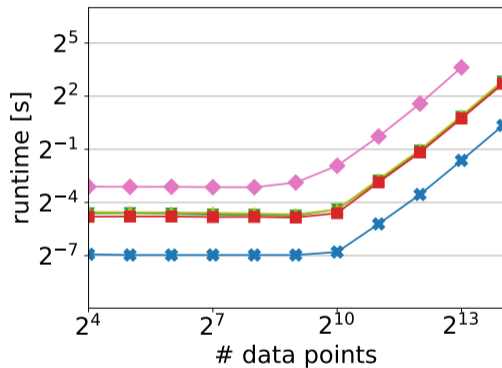
			memory	bandwidth	FP64
GPUs	NVIDIA	A100	40 GB HBM2e	1555 GB/s	9.7 TFLOPS
		P100	16 GB HBM2	732 GB/s	4.7 TFLOPS
		RTX 3080	10 GB GDDR6X	760.3 GB/s	465.1 GFLOPS
		GTX 1080 Ti	11 GB GDDR5X	484.4 GB/s	354.4 GFLOPS
	AMD Radeon Pro VII		16 GB HBM2	1024 GB/s	6.5 TFLOPS
	Intel	UHD P630	53.8 GB DDR4	41.6 GB/s	96 GFLOPS
		Iris Xe MAX	4 GB LPDDR4x	68 GB/s	emulated
			base/boost freq.	bandwidth	# cores/# HT
CPUs	AMD	EPYC 7742	2.25/3.4 GHz	204.8 GB/s	2 · 64/2 · 128
		Ryzen TR 3960X	3.8/4.5 GHz	102.4 GB/s	24/48
	Intel	Xeon Phi 7210	1.3/1.5 GHz	102 GB/s	64/256
		Xeon E-2176G	3.7/4.7 GHz	41.6 GB/s	6/12
		Core i9-10920X	3.5/4.6 GHz	94 GB/s	12/24

NVIDIA GPUs: A100 vs. RTX 3080 - 4096 features

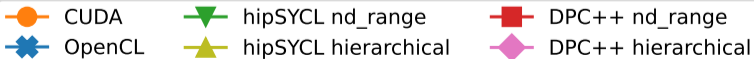
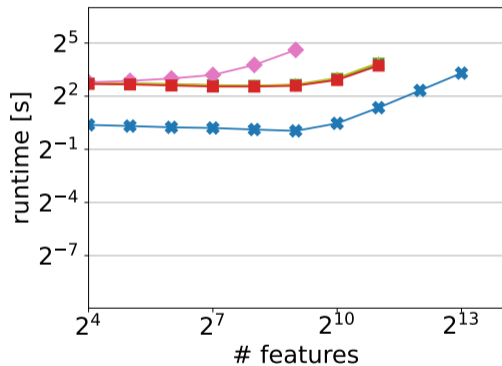


AMD GPU: Radeon Pro VII

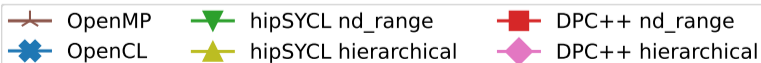
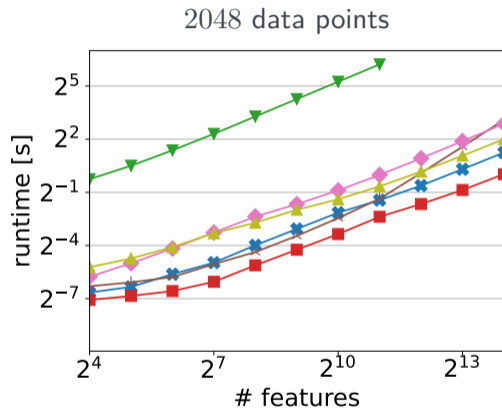
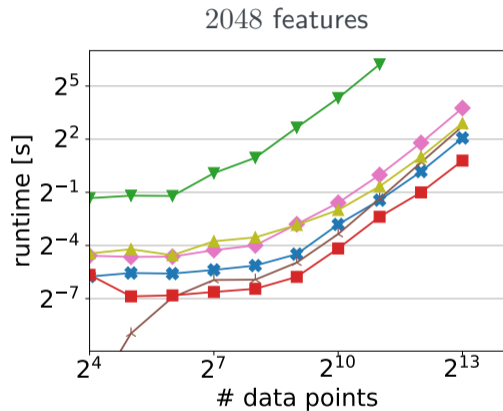
4096 features



32 768 data points



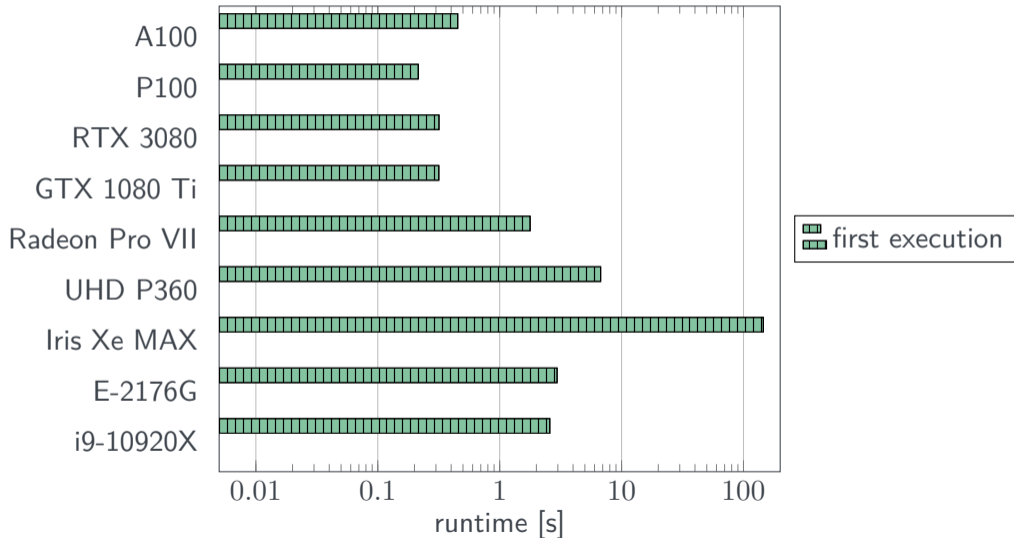
Intel CPU: Core i9-10920X



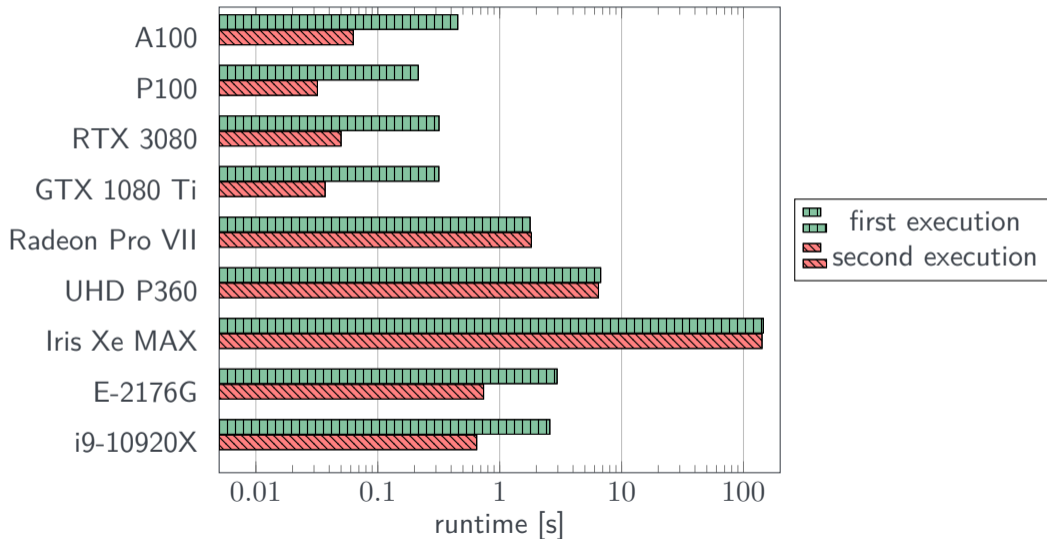
Additional Observations

- results for the P100 and GTX 1080 Ti nearly identical to the A100 and RTX 3080 respectively
- overall behavior the same on Intel GPUs
- OpenCL faster than DPC++ on the Iris Xe MAX GPU
- overall behavior on CPUs nearly identical (except OpenMP)
- on every hardware: DPC++ hierarchical slower than `nd_range`

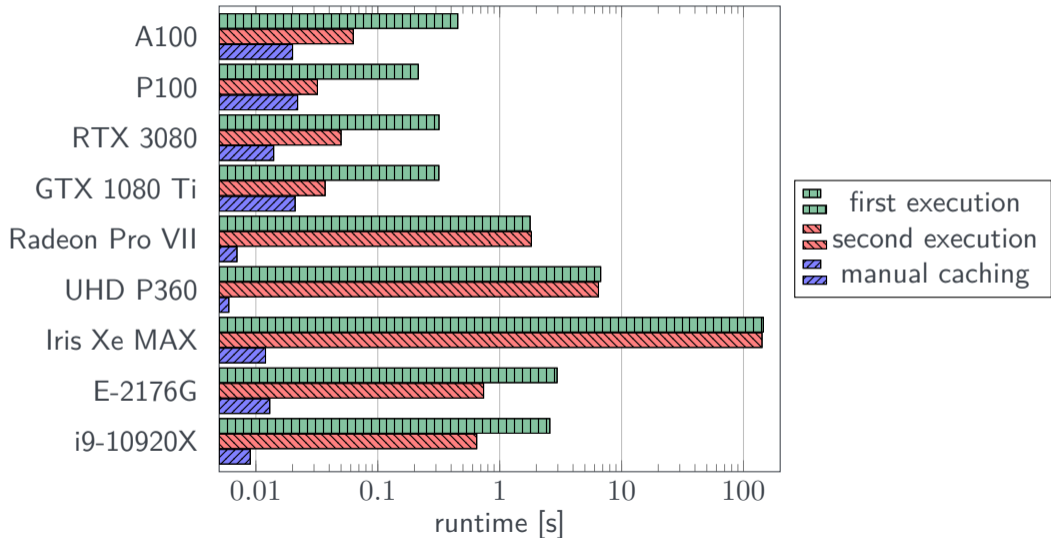
OpenCL JIT Compilation Overhead



OpenCL JIT Compilation Overhead



OpenCL JIT Compilation Overhead



Conclusion

4

Conclusion - Contribution

- Open Source *Parallel Least Squares Support Vector Machine* (PLSSVM)
 - multiple backends: OpenMP, CUDA, OpenCL, SYCL
 - be able to target GPUs from NVIDIA, AMD, and Intel as well as CPUs



<https://github.com/SC-SGS/PLSSVM>

Conclusion - Contribution

- Open Source *Parallel Least Squares Support Vector Machine* (PLSSVM)
 - multiple backends: OpenMP, CUDA, OpenCL, SYCL
 - be able to target GPUs from NVIDIA, AMD, and Intel as well as CPUs
- comparison of a standard problem (matrix-vector multiplication in the CG algorithm) using different programming frameworks on different hardware platforms
- based on our findings: recommendation of which framework to use when



<https://github.com/SC-SGS/PLSSVM>

Conclusion - Results



see our paper
for more results

NVIDIA GPUs
only?

Conclusion - Results



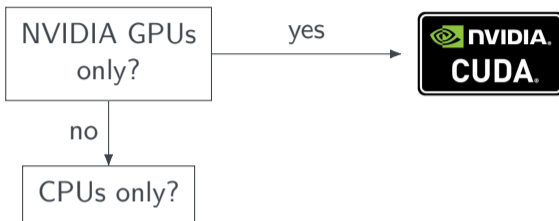
see our paper
for more results



Conclusion - Results



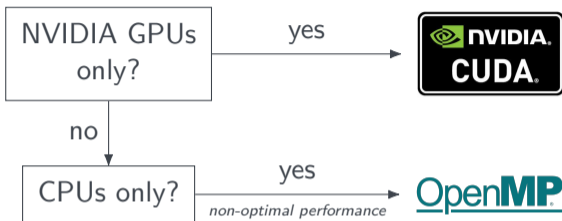
see our paper
for more results



Conclusion - Results



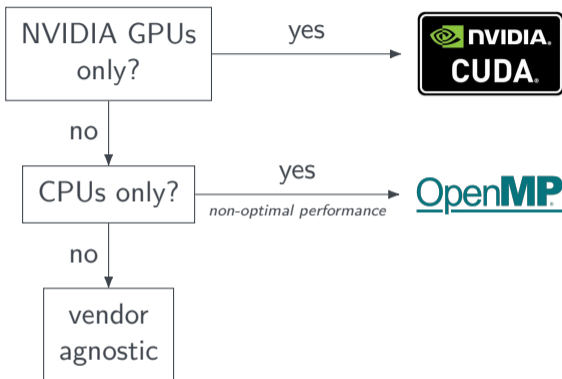
see our paper
for more results



Conclusion - Results



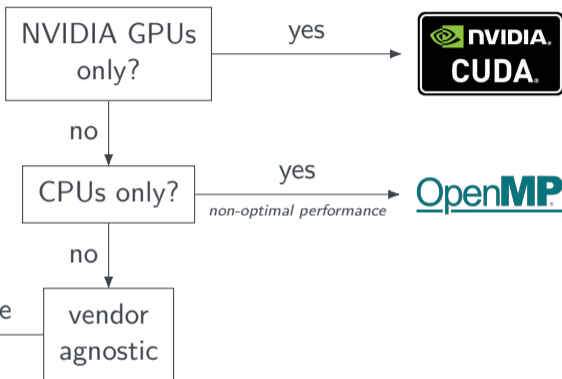
see our paper
for more results



Conclusion - Results



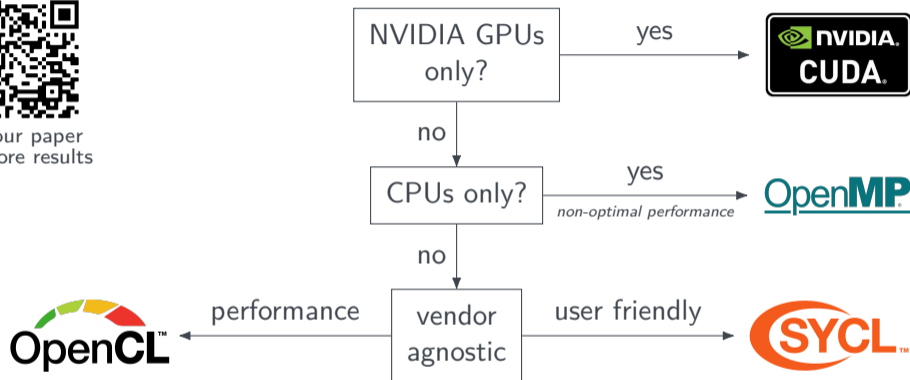
see our paper
for more results



Conclusion - Results



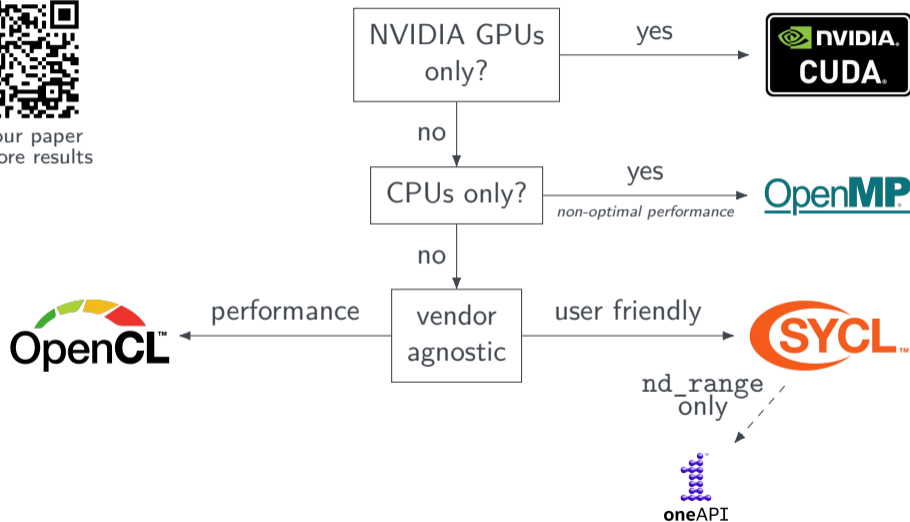
see our paper
for more results



Conclusion - Results



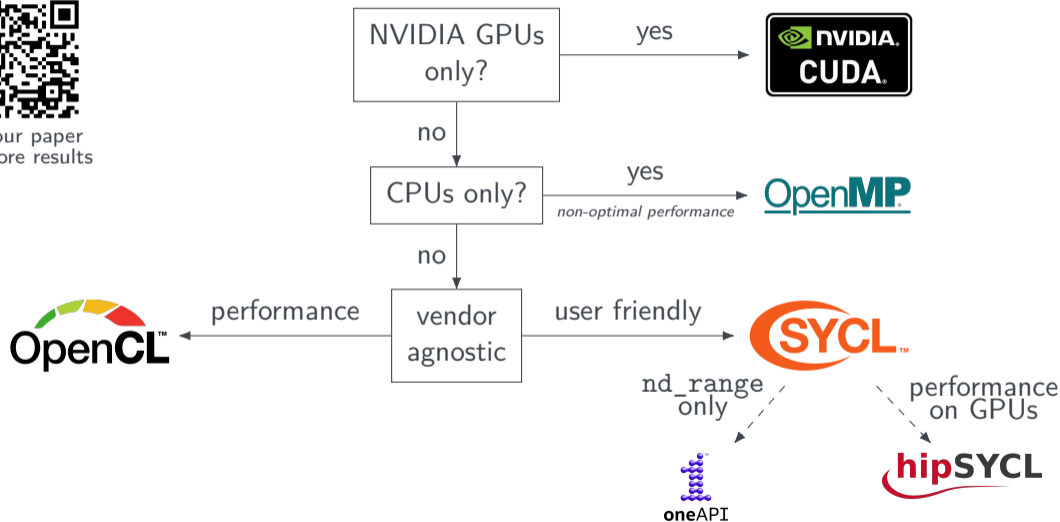
see our paper
for more results



Conclusion - Results



see our paper
for more results



Future Work

- further optimizing the OpenMP backend
- add additional backends (e.g., Kokkos or OpenMP target offloading)
- investigate other SYCL implementations, e.g., ComputeCpp
- investigate performance on other hardware platforms, e.g., FPGAs

Future Work

- further optimizing the OpenMP backend
- add additional backends (e.g., Kokkos or OpenMP target offloading)
- investigate other SYCL implementations, e.g., ComputeCpp
- investigate performance on other hardware platforms, e.g., FPGAs

Current Work in Progress

- sparse (CG) implementation
- support for distributed systems and multi-node execution via MPI
- investigate mixed precision and the usage of special ML hardware (e.g., NVIDIA's tensor cores)



University of Stuttgart
Germany



Marcel Breyer 

Marcel.Breyer@ipvs.uni-
stuttgart.de



Alexander Van Craen 

Alexander.Van-Craen@ipvs.uni-
stuttgart.de



Dirk Pflüger 

Dirk.Pflueger@ipvs.uni-
stuttgart.de