Performance Evaluation and Improvements of the *PoCL* Open-Source OpenCL Implementation on Intel CPUs

Tobias Baumann, Matthias Noack, Thomas Steinke

Supercomputing Department
Zuse Institute Berlin

9th International Workshop on OpenCL and SYCL, 26-29 April 2021

Motivation

- Top500 status quo: **91.8 %** of all systems feature **Intel CPUs**

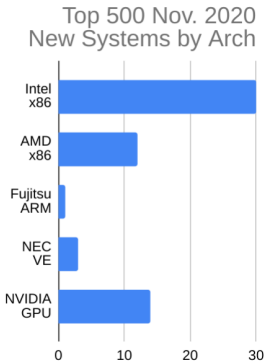
TOP500 November 2020. https://top500.org/lists/top500/2020/11/

Motivation

- Top500 status quo: **91.8 %** of all systems feature **Intel CPUs**

- Top500 trends: increased **diversity**
  - accelerators and co-processors (27.2 %)
  - ARM CPUs (1 %, including #1)
  - x86 CPUs from AMD (4.4 %)

⇒ **portability** more important than ever



TOP500 November 2020. https://top500.org/lists/top500/2020/11/

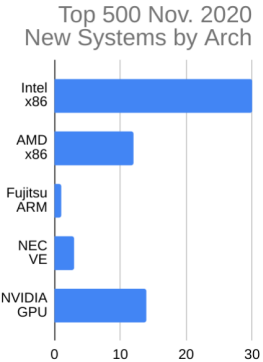
Motivation

- Top500 status quo: **91.8 %** of all systems feature **Intel CPUs**

- Top500 trends: increased **diversity**
  - accelerators and co-processors (27.2 %)
  - ARM CPUs (1 %, including #1)
  - x86 CPUs from AMD (4.4 %)

⇒ **portability** more important than ever

- **OpenCL** widely supported, but not every vendors 1st choice
  ⇒ vendor-independent **open source** impl.: **PoCL**



TOP500 November 2020. https://top500.org/lists/top500/2020/11/

Portable Computing Language (PoCL)

- portable, open source (MIT) OpenCL implementation
  $\Rightarrow$ maintained by Customized Parallel Computing group at Tampere University, Finland

Jääskeläinen, P., De La Lama, C. S., Schnetter, E., Raiskila, K., Takala, J., Berg, H. (2015). pocl: A Performance-Portable OpenCL Implementation. International Journal of Parallel Programming, (43)5, 752–785. https://doi.org/10.1007/s10766-014-0320-y

Portable Computing Language (PoCL)

- portable, open source (MIT) OpenCL implementation
  - $\Rightarrow$ maintained by Customized Parallel Computing group at Tampere University, Finland
- aims to support a variety of compute devices in a single platform
- longer term goal: enhance performance portability of OpenCL programs

Jääskeläinen, P., De La Lama, C. S., Schnetter, E., Raiskila, K., Takala, J., Berg, H. (2015). pocl: A Performance-Portable OpenCL Implementation. International Journal of Parallel Programming, (43)5, 752–785. https://doi.org/10.1007/s10766-014-0320-y

Portable Computing Language (PoCL)

- portable, open source (MIT) OpenCL implementation
  - $\Rightarrow$ maintained by Customized Parallel Computing group at Tampere University, Finland
- aims to support a variety of compute devices in a single platform
- longer term goal: enhance performance portability of OpenCL programs
- currently supported devices:
  - various CPUs
  - NVIDIA GPUs via libcuda
  - HSA-supported GPUs
  - TCE ASIPs (experimental)
  - multiple (private) adaptations in active production use

Jääskeläinen, P., De La Lama, C. S., Schnetter, E., Raiskila, K., Takala, J., Berg, H. (2015). pocl: A Performance-Portable OpenCL Implementation. International Journal of Parallel Programming, (43)5, 752–785. https://doi.org/10.1007/s10766-014-0320-y

Portable Computing Language (PoCL)

- portable, open source (MIT) OpenCL implementation
  - $\Rightarrow$ maintained by Customized Parallel Computing group at Tampere University, Finland
- aims to support a variety of compute devices in a single platform
- longer term goal: enhance performance portability of OpenCL programs
- currently supported devices:
  - various CPUs
  - NVIDIA GPUs via libcuda
  - HSA-supported GPUs
  - TCE ASIPs (experimental)
  - multiple (private) adaptations in active production use
- compiler: Clang/LLVM
- vectorised math libraries: SLEEF, libclc

Jääskeläinen, P., De La Lama, C. S., Schnetter, E., Raiskila, K., Takala, J., Berg, H. (2015). pocl: A Performance-Portable OpenCL Implementation. International Journal of Parallel Programming, (43)5, 752–785. https://doi.org/10.1007/s10766-014-0320-y

Contributions

1. Performance evaluation PoCL vs. Intel OpenCL

Contributions

1. Performance evaluation PoCL vs. Intel OpenCL

2. PoCL bug fixes:
   - WG size algorithm (one-off)
   - scheduler (integer division)

Contributions

1. Performance evaluation PoCL vs. Intel OpenCL

2. PoCL bug fixes:
   - WG size algorithm (one-off)
   - scheduler (integer division)

3. New PoCL CPU driver using Intel Threading Building Blocks (TBB)

Contributions

1. Performance evaluation PoCL vs. Intel OpenCL

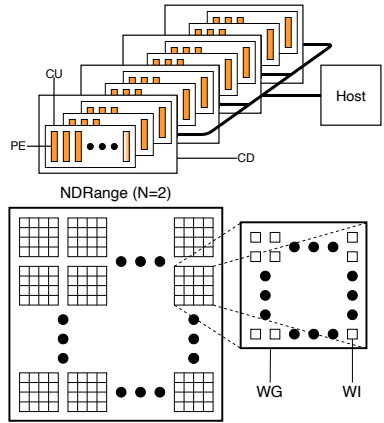
2. PoCL bug fixes:
   - WG size algorithm (one-off)
   - scheduler (integer division)

3. New PoCL CPU driver using Intel Threading Building Blocks (TBB)

4. Evaluation of vectorisation strategies and LLVM's loop and VPlan vectorisers

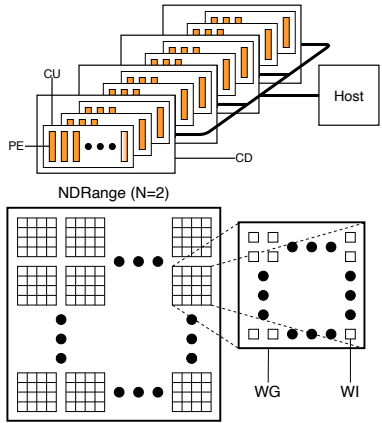
PoCL Work-Group Size Algorithm (simplified, for CPUs)

- Problem: How to determine WG size if none is specified?



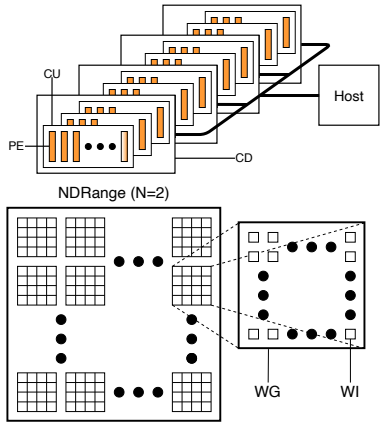
PoCL Work-Group Size Algorithm (simplified, for CPUs)

- Problem: How to determine WG size if none is specified?
- Step 1: **maximise** WG size
  - maximum WG size: 4096 WI
  - preferred WG multiple (for vectorisation): 8 WI



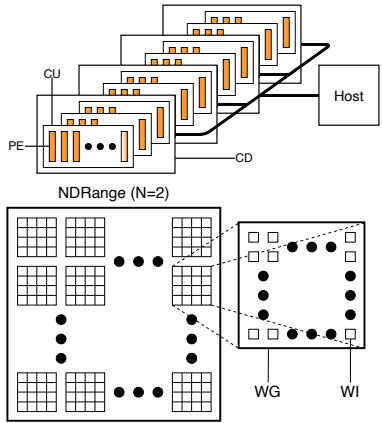
PoCL Work-Group Size Algorithm (simplified, for CPUs)

- Problem: How to determine WG size if none is specified?
- Step 1: **maximise** WG size
  - maximum WG size: 4096 WI
  - preferred WG multiple (for vectorisation): 8 WI
- Step 2: **reduce** WG size
  - reason: at least one WG/CU
  - minimum WG size: 32 WI
  - can lead to distributions like 1.6 WGs/CU



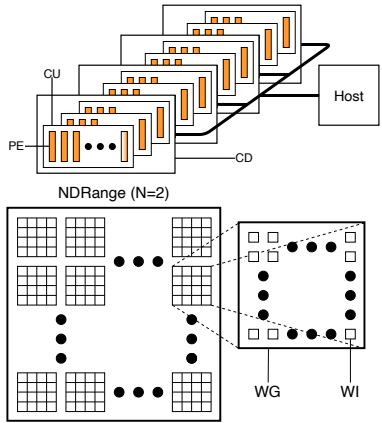
PoCL Work-Group Size Algorithm (simplified, for CPUs)

- Problem: How to determine WG size if none is specified?
- Step 1: **maximise** WG size
  - maximum WG size: 4096 WI
  - preferred WG multiple (for vectorisation): 8 WI
- Step 2: **reduce** WG size
  - reason: at least one WG/CU
  - minimum WG size: 32 WI
  - can lead to distributions like 1.6 WGs/CU
- parallelisation threshold: $\#CU \times 32$ WI



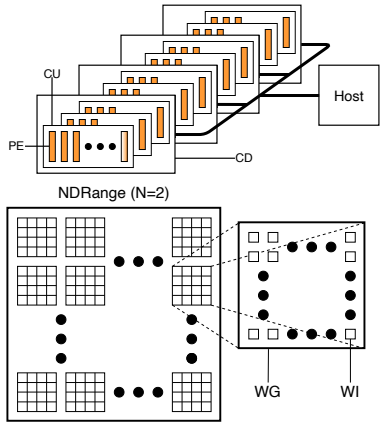
PoCL Work-Group Size Algorithm (simplified, for CPUs)

- Problem: How to determine WG size if none is specified?
- Step 1: **maximise** WG size
  - maximum WG size: 4096 WI
  - preferred WG multiple (for vectorisation): 8 WI
- Step 2: **reduce** WG size
  - reason: at least one WG/CU
  - minimum WG size: 32 WI
  - can lead to distributions like 1.6 WGs/CU
- parallelisation threshold: $\#CU \times 32$ WI
- omit whole algorithm: explicitly specify WG size



PoCL Work-Group Size Algorithm (simplified, for CPUs)

- Problem: How to determine WG size if none is specified?
- Step 1: **maximise** WG size
  - maximum WG size: 4096 WI
  - preferred WG multiple (for vectorisation): 8 WI
- Step 2: **reduce** WG size
  - reason: at least one WG/CU
  - minimum WG size: 32 WI
  - can lead to distributions like 1.6 WGs/CU
- parallelisation threshold: $\#CU \times 32$ WI
- omit whole algorithm: explicitly specify WG size



Improvement #1: minor off-by-one issue in step 2

PoCL Scheduling Algorithm (*pthread* device)

- one (software) thread per CU (hardware thread)
- problem: schedule WGs to threads

PoCL Scheduling Algorithm (*pthread* device)

- one (software) thread per CU (hardware thread)
- problem: schedule WGs to threads
- pull scheduler
- threads pass through the scheduler **sequentially**
- algorithm: number of remaining WGs $\div$ #threads

PoCL Scheduling Algorithm (*pthread* device)

- one (software) thread per CU (hardware thread)
- problem: schedule WGs to threads
- pull scheduler
- threads pass through the scheduler **sequentially**
- algorithm: number of remaining WGs $\div$ #threads

- example: schedule 64 WGs to 4 threads
  - balanced workload
  - no scheduling overhead

PoCL Scheduling Algorithm Balanced Workload

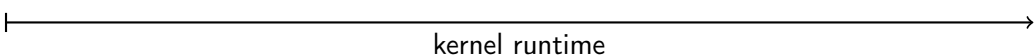
WGs remaining: 64 WGs scheduled: 0 WGs to schedule: 16

t_1:

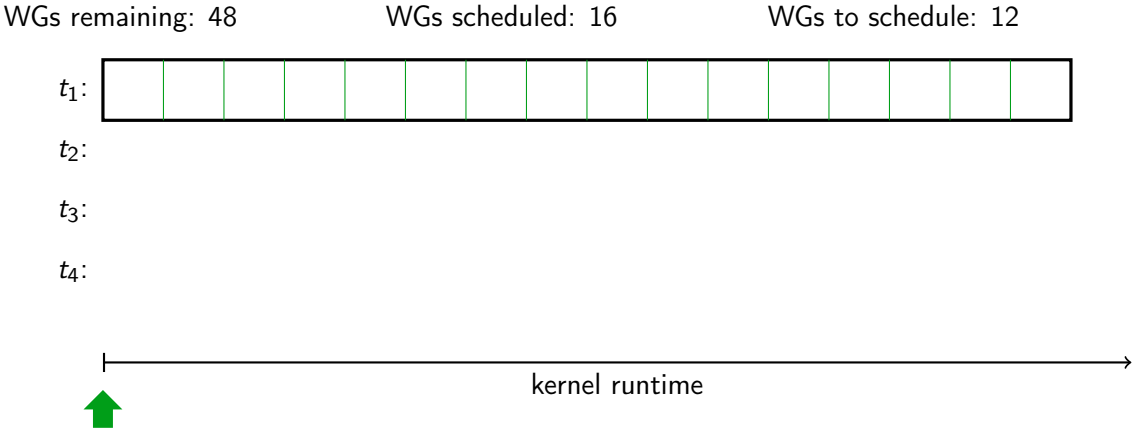
t_2:

t_3:

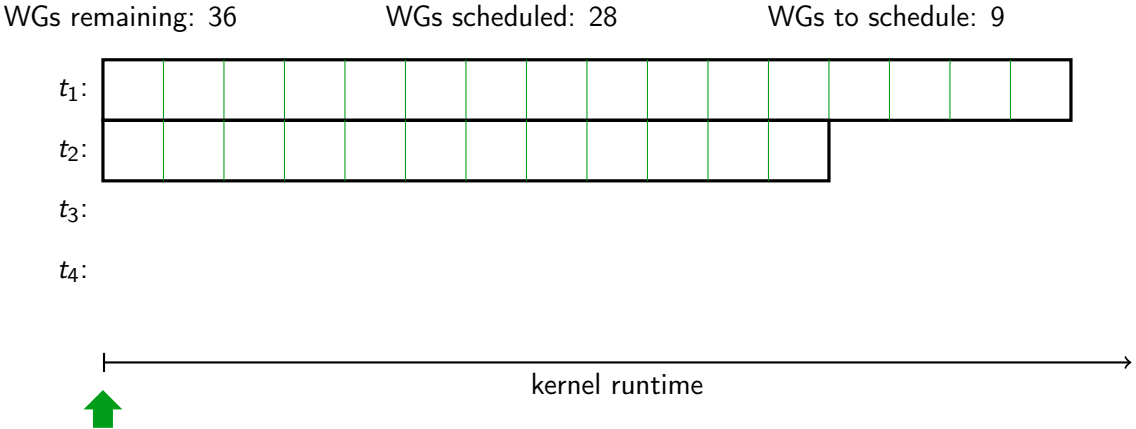
t_4:



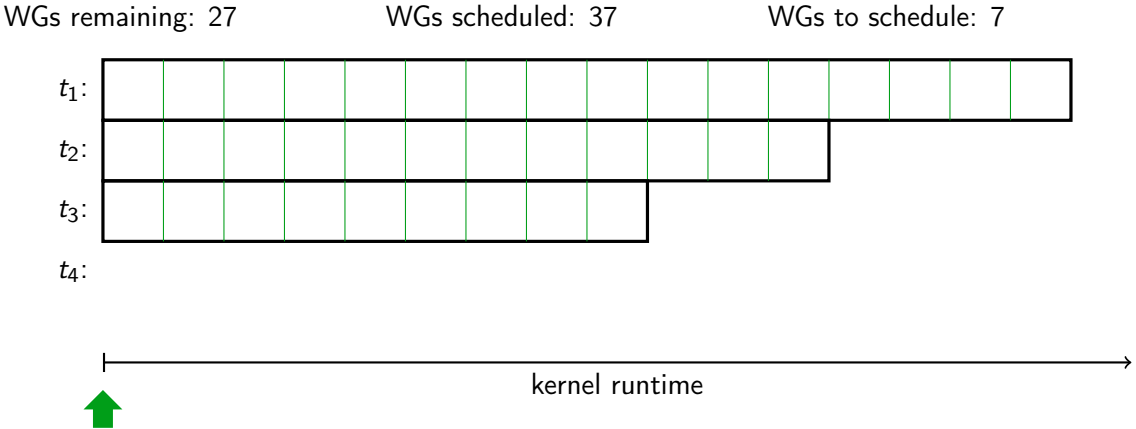
PoCL Scheduling Algorithm Balanced Workload



PoCL Scheduling Algorithm Balanced Workload

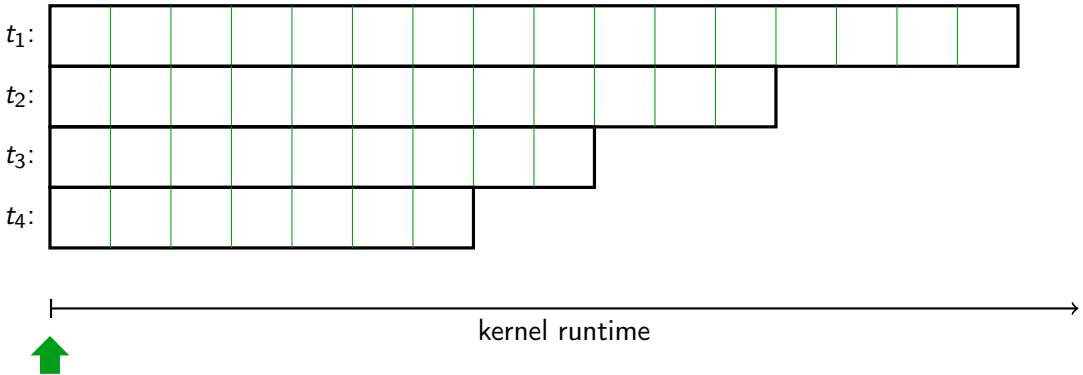


PoCL Scheduling Algorithm Balanced Workload

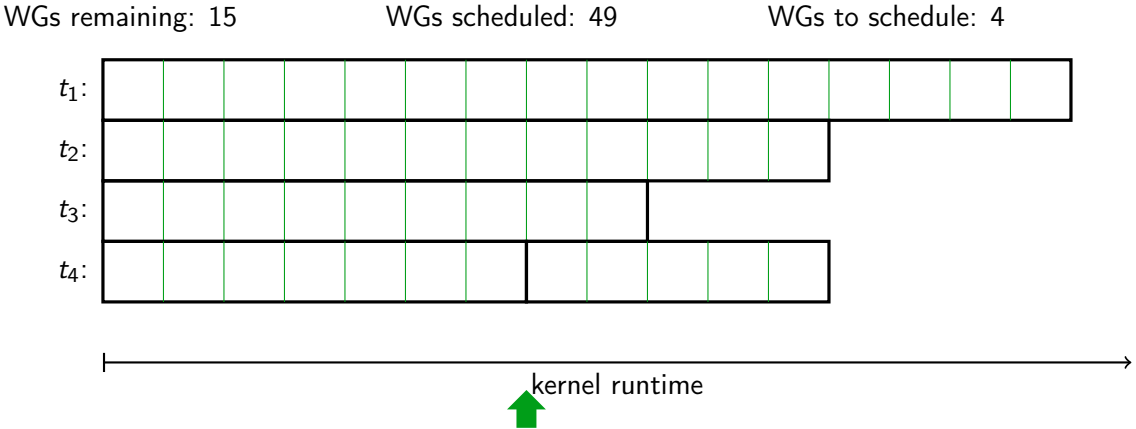


PoCL Scheduling Algorithm Balanced Workload

WGs remaining: 20 WGs scheduled: 44 WGs to schedule: 5

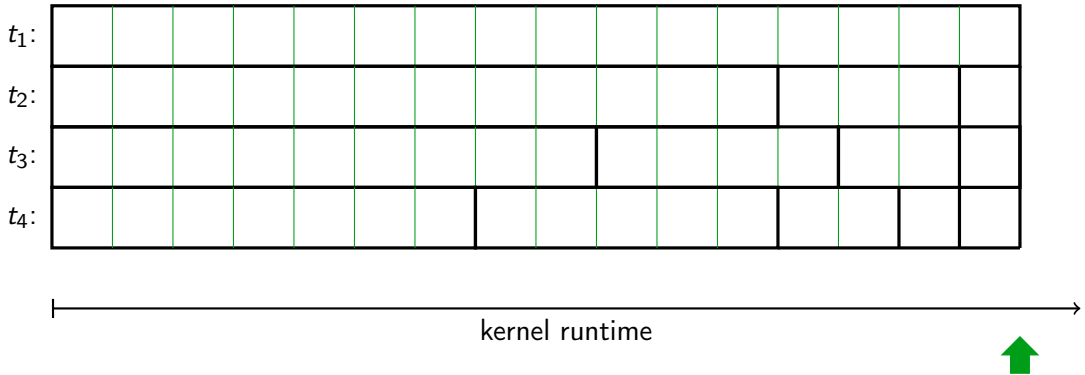


PoCL Scheduling Algorithm Balanced Workload

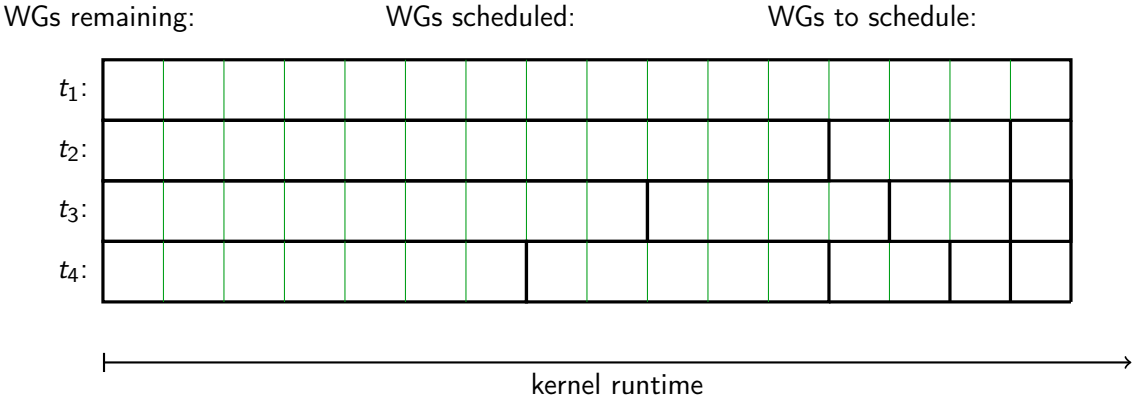


PoCL Scheduling Algorithm Balanced Workload

WGs remaining: 0 WGs scheduled: 64 WGs to schedule: -

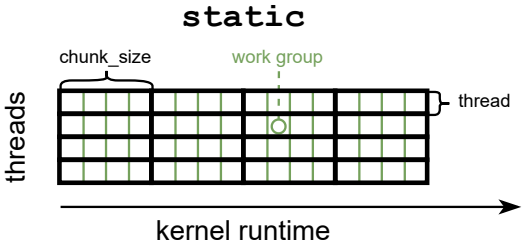


PoCL Scheduling Algorithm Balanced Workload

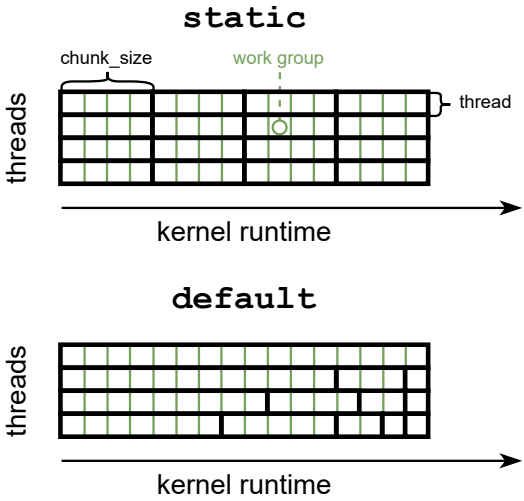


Improvement #2: integer division bug

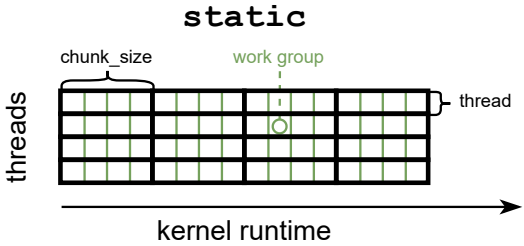
Scheduling Algorithms: OpenMP "static" vs. PoCL "default"

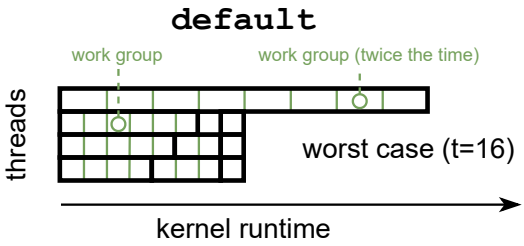


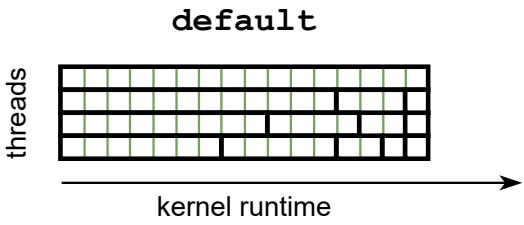
Scheduling Algorithms: OpenMP "static" vs. PoCL "default"



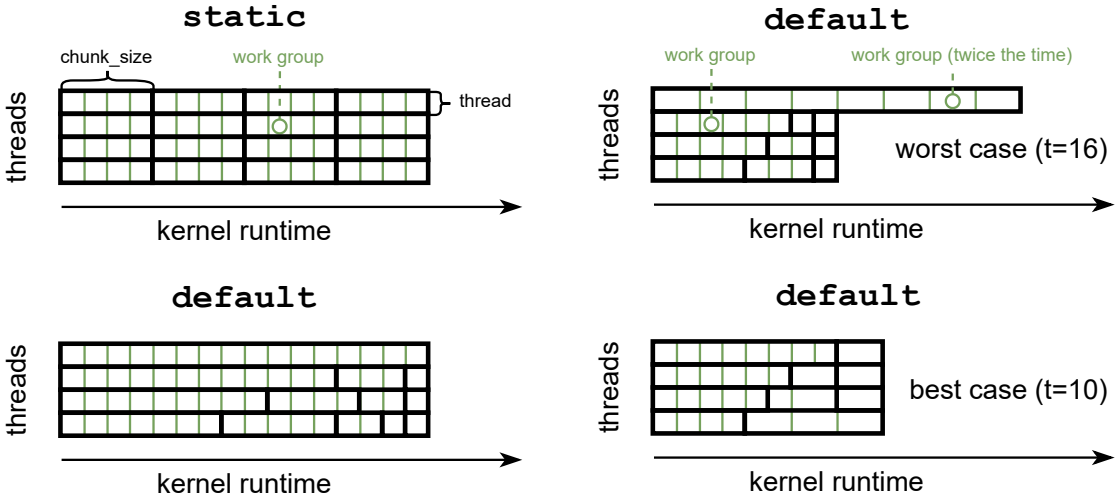
Scheduling Algorithms: OpenMP "static" vs. PoCL "default"







Scheduling Algorithms: OpenMP "static" vs. PoCL "default"



TBB device for PoCL

- Intel Threading Building Blocks (TBB) library
  - higher level alternative to *pthread*
  - used by Intel OpenCL
  - open source (Apache 2.0)
  - available **partitioners** for scheduling: auto, affinity, static, simple
  - optional parameter for all partitioners: grain size
  - additional work stealing strategy

TBB device for PoCL

- Intel Threading Building Blocks (TBB) library
  - higher level alternative to *pthread*
  - used by Intel OpenCL
  - open source (Apache 2.0)
  - available **partitioners** for scheduling: auto, affinity, static, simple
  - optional parameter for all partitioners: grain size
  - additional work stealing strategy
- Integration into PoCL:
  - derived from the *pthread* device
  - replaced relevant code sections with calls to TBB library
  - custom env vars: `POCL_TBB_PARTITIONER`, `POCL_TBB_GRAIN_SIZE`

Setup

- Software:
  - Intel CPU Runtime for OpenCL Applications 2021.1
  - PoCL 1.4 LLVM 9
  - PoCL 1.6 LLVM 11
  - PoCL 1.6 LLVM 11 with proposed TBB device

- Hardware: dual socket Intel Xeon Gold 6138 (Skylake)
  - 20 cores per socket $\Rightarrow$ 80 hardware threads
  - AVX-512 units: 2 per core $\Rightarrow$ 1 per hardware thread

Synthetic Micro Benchmark: op

- synthetic, low-level, micro benchmark
- goal: measure the performance of **operators** and OpenCL **built-in functions**

```
void op_kernel (DATA_T* in_a, DATA_T* in_b, DATA_T* out)
{
    int id = get_global_id(0);
    DATA_T tmp_a = in_a[id];
    DATA_T tmp_b = in_b[id];
    for (int i = 0; i < ITERATIONS; ++i)
    {
        tmp_a = OP_TO_BENCHMARK (tmp_a, tmp_b); //data dep.
    }
    out[id] = tmp_a;
}
```

- compile-time specialisation via: `DATA_T`, `ITERATIONS`, and `OP_TO_BENCHMARK`

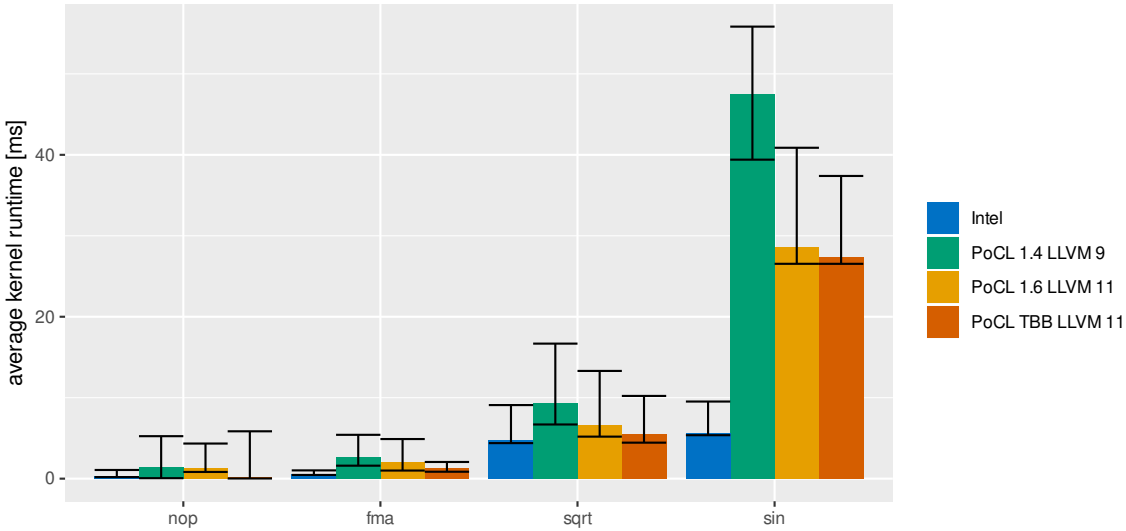
Automatic vs. Manual Vectorisation

- **automatically** by the compiler or **manually** by using vector data types
- $\#WIs = \#data\_elements \div vector\_length$

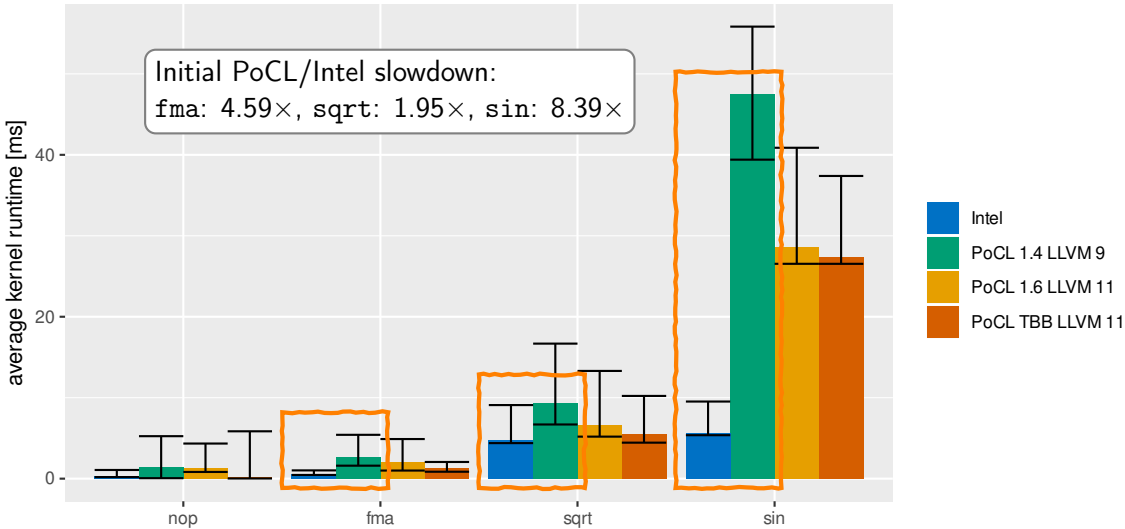
- e.g. resulting ■ work-items (WIs) spanning 16 □ data elements:

vec. mode	vec. length	#WIs	logical data layout
automatic	1	16	
manual4	4	4	
manual8	8	2	

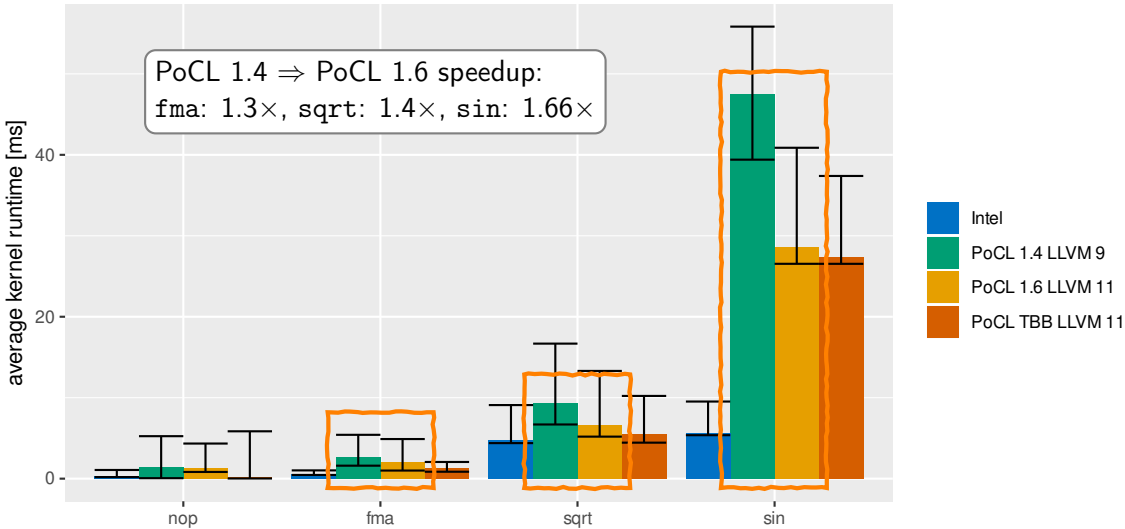
Synthetic Micro Benchmark: op



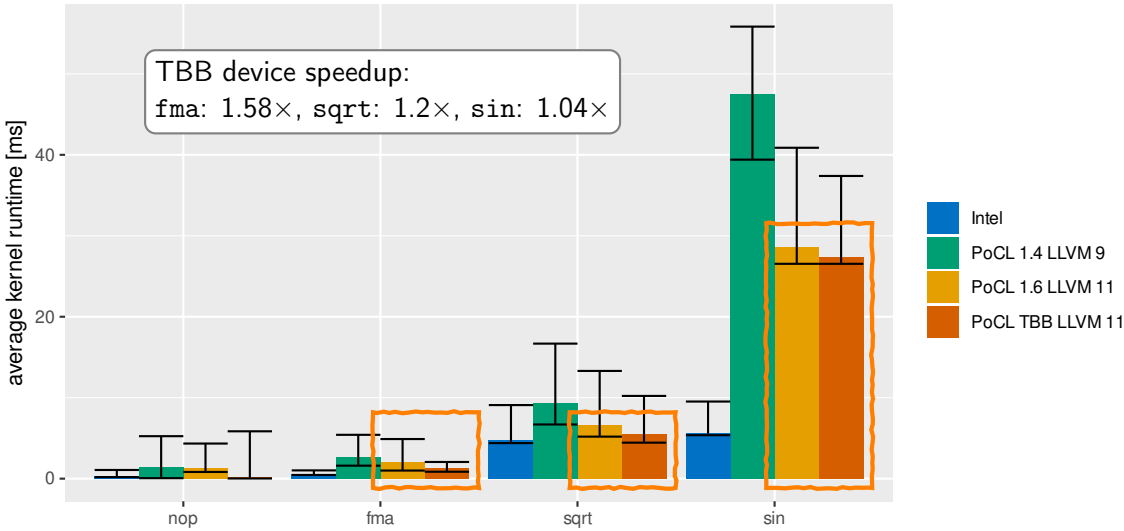
Synthetic Micro Benchmark: op



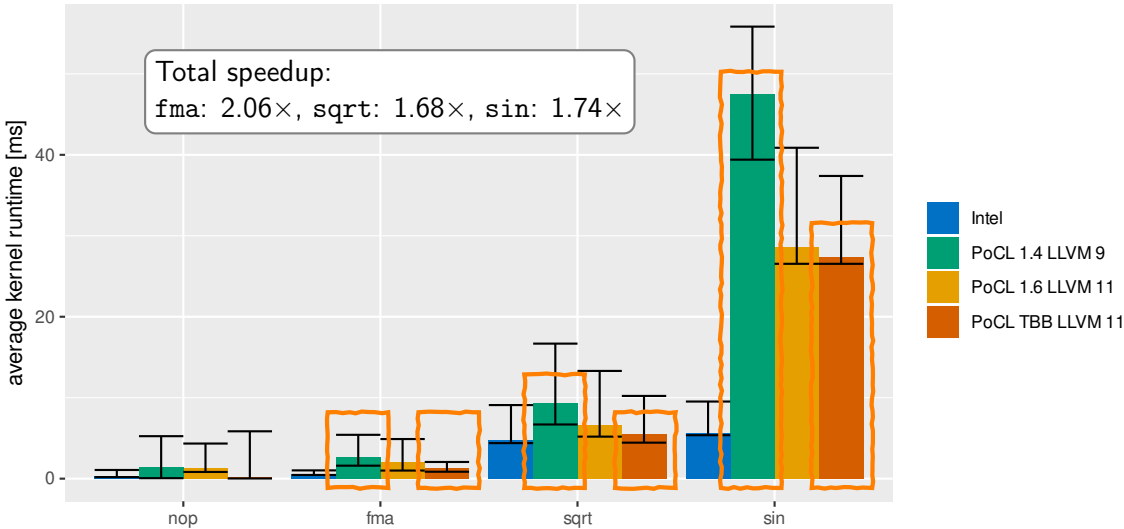
Synthetic Micro Benchmark: op



Synthetic Micro Benchmark: op

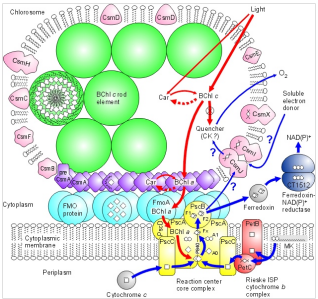


Synthetic Micro Benchmark: op



Proxy Application Benchmark: hexciton

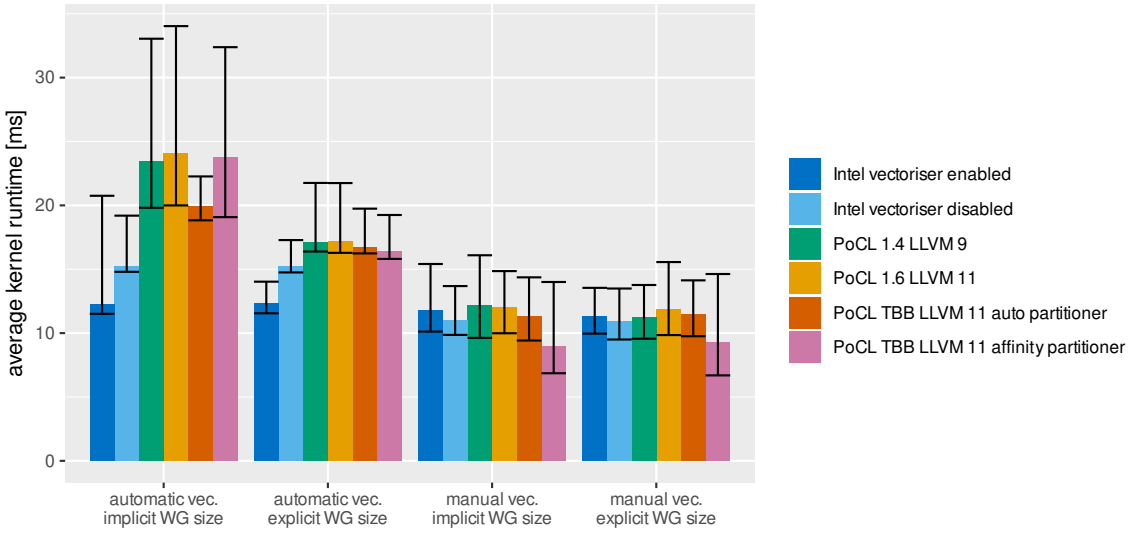
- proxy application benchmark for the **DM-HEOM** code
- same kernel in different variants and stages of optimisation
- vectorisation strategies: **automatic** and **manual8**
- OpenMP version using KART library for runtime compilation
- instruction mix: streamlined FMA instructions
- fixed number of 512×1024 small 7×7 matrices



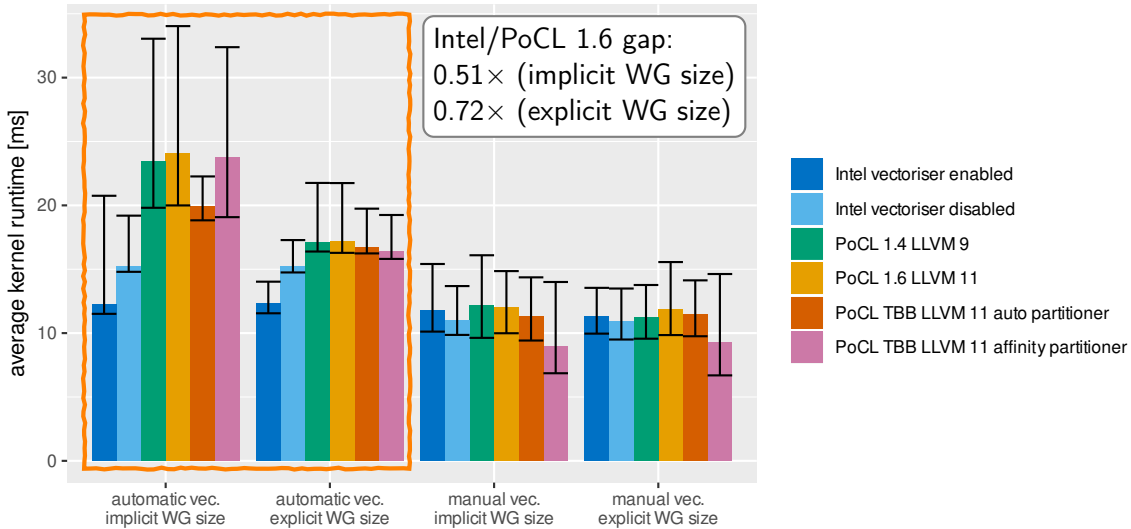
[Image by University of Copenhagen Biology Department]

Matthias Noack, Alexander Reinefeld, Tobias Kramer, and Thomas Steinke. 2018. DM-HEOM: A Portable and Scalable Solver-Framework for the Hierarchical Equations of Motion. In 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). 947–956. https://doi.org/10.1109/IPDPSW.2018.00149

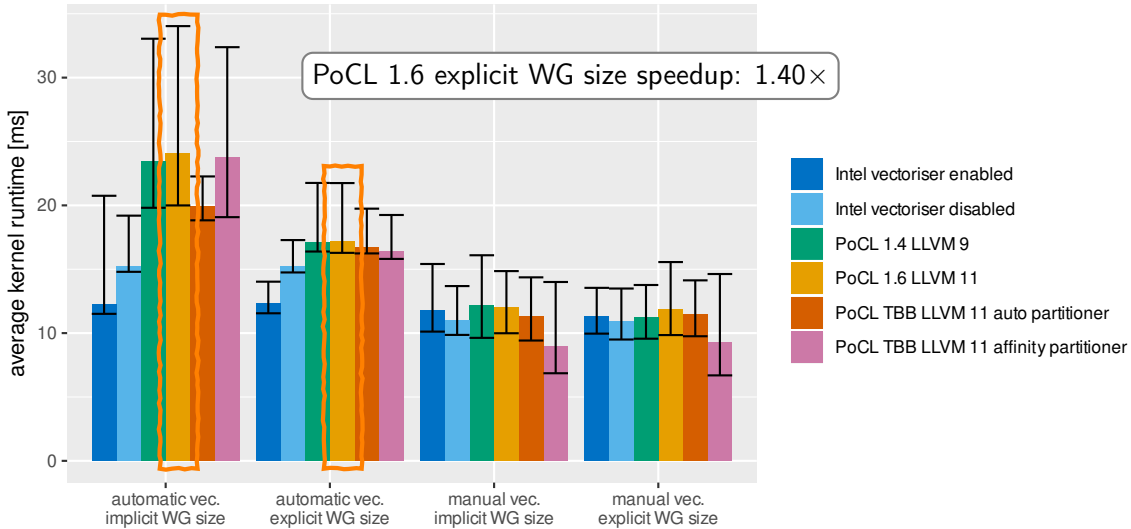
Proxy Application Benchmark: hexciton



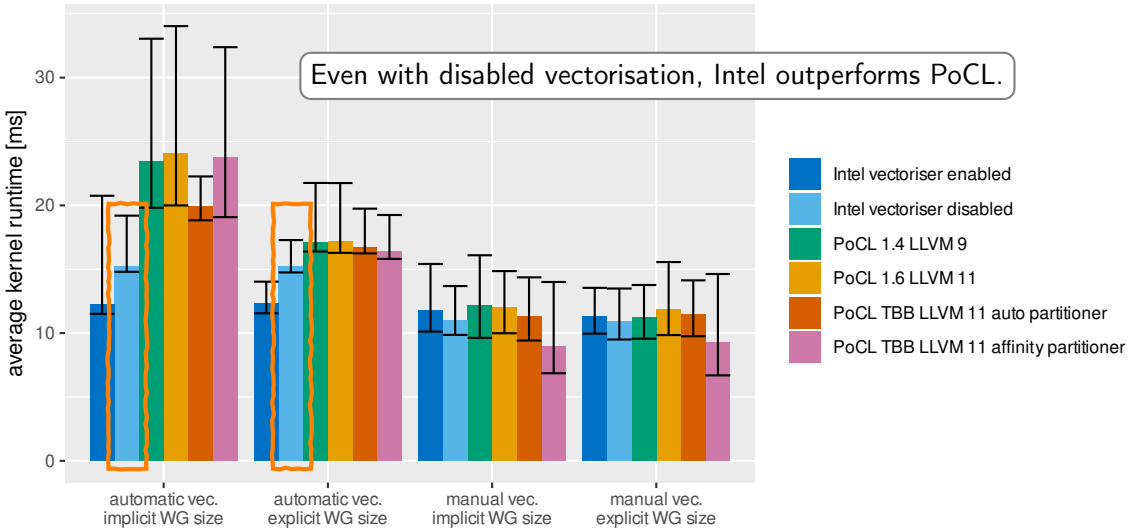
Proxy Application Benchmark: hexciton



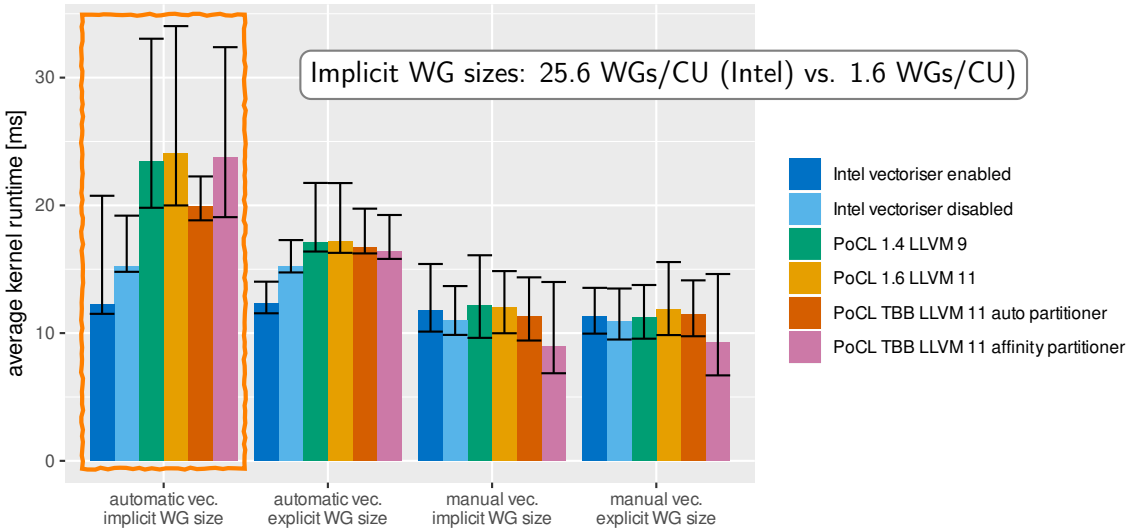
Proxy Application Benchmark: hexciton



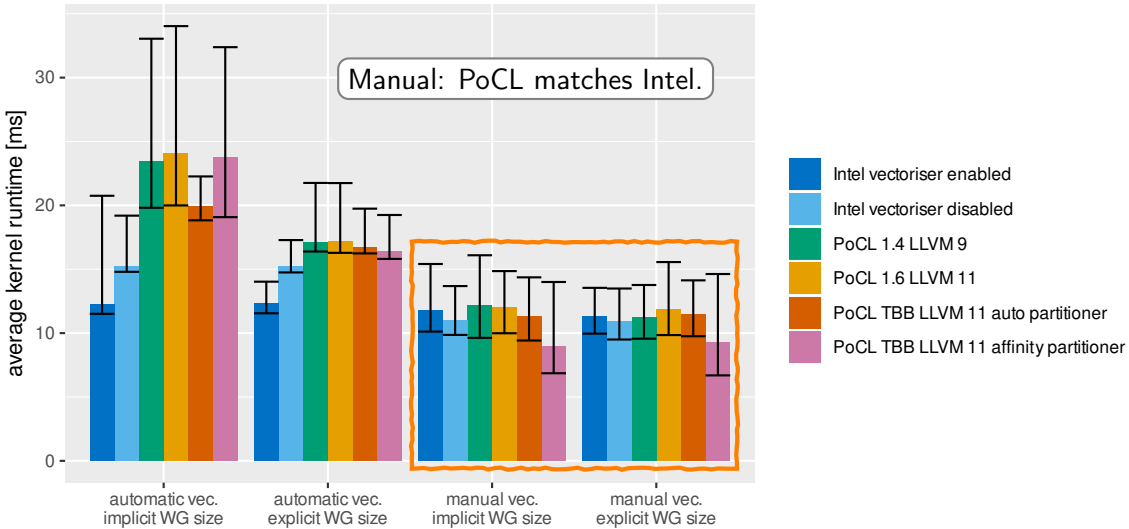
Proxy Application Benchmark: hexciton



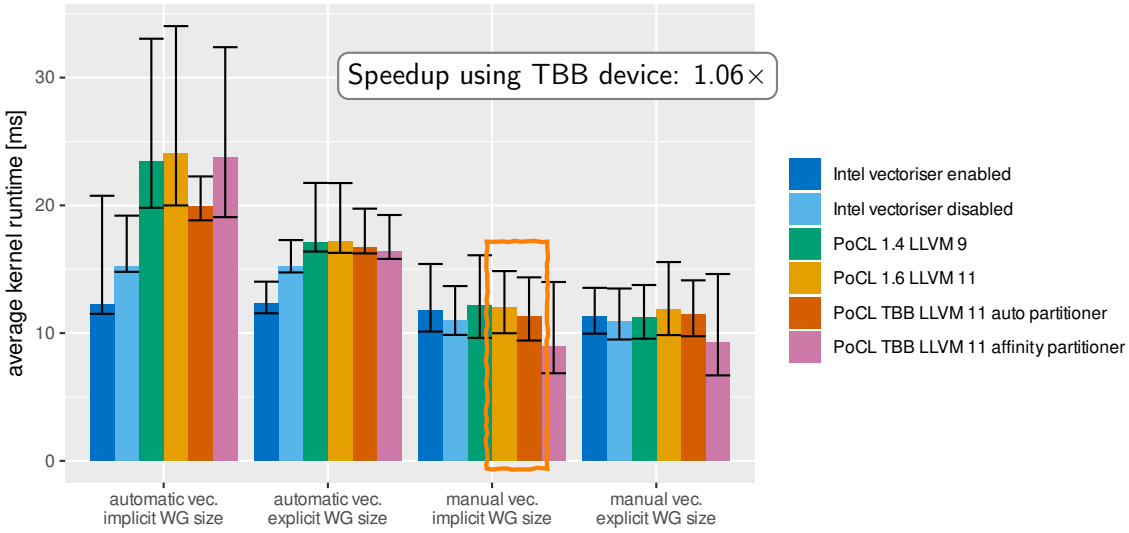
Proxy Application Benchmark: hexciton



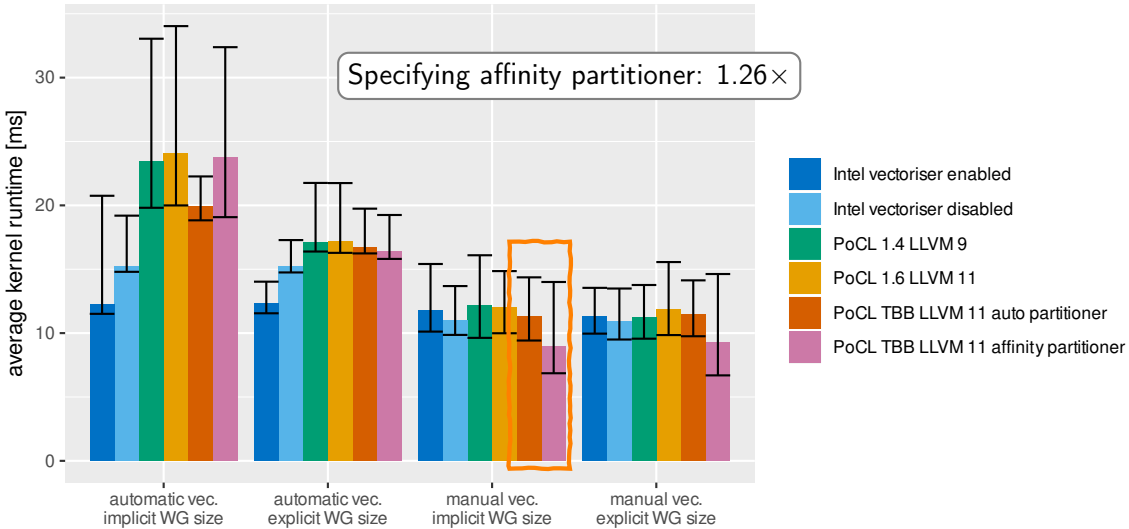
Proxy Application Benchmark: hexciton



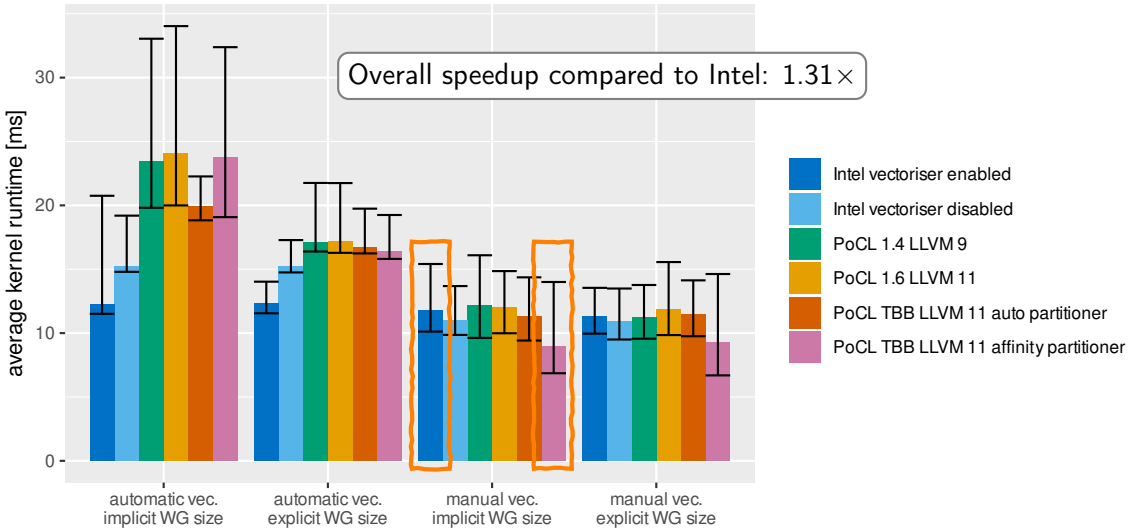
Proxy Application Benchmark: hexciton



Proxy Application Benchmark: hexciton

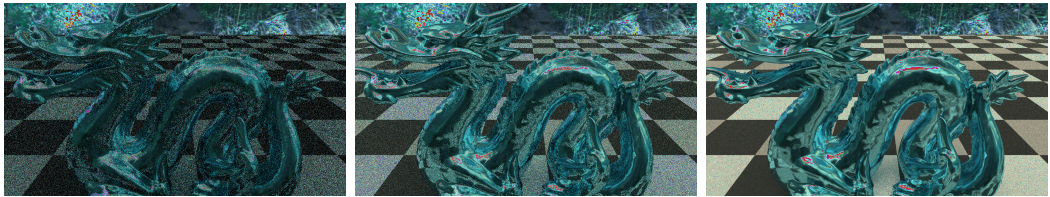


Proxy Application Benchmark: hexciton



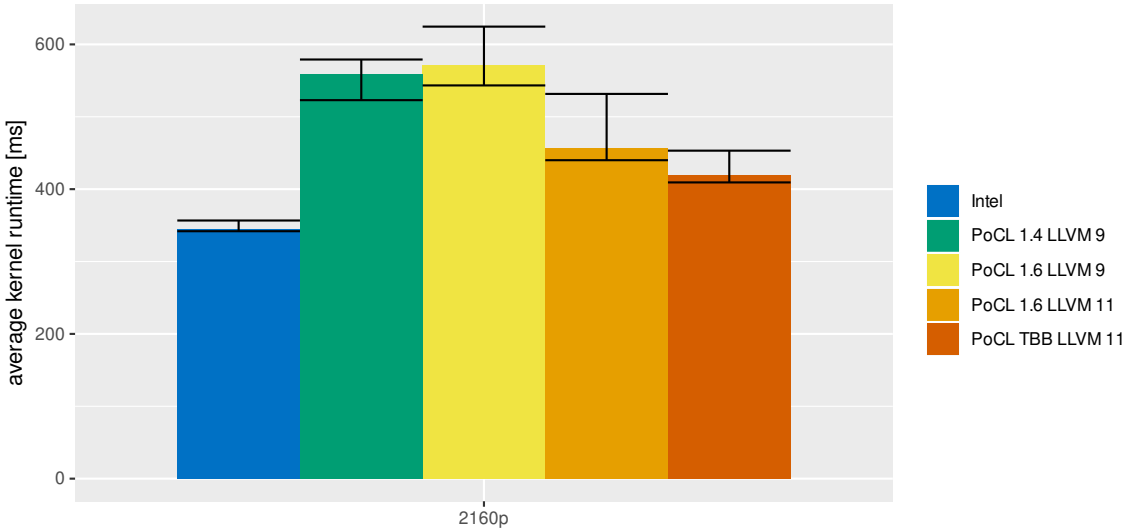
Real World Application Benchmark: raytracing

- derived from a raytracing code published on GitHub
  - $\Rightarrow$ reflect average real world application code
- more complex data structures
- work-items can take different code paths
- contains trigonometric and various other built-in functions

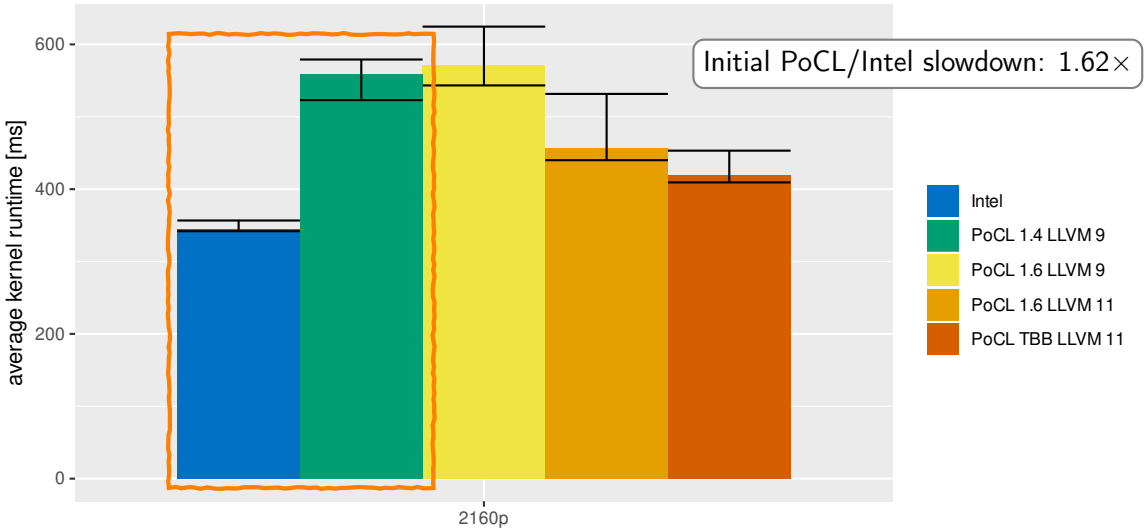


Raytracing benchmark source code on Github. https://github.com/new2f7/RayTracing

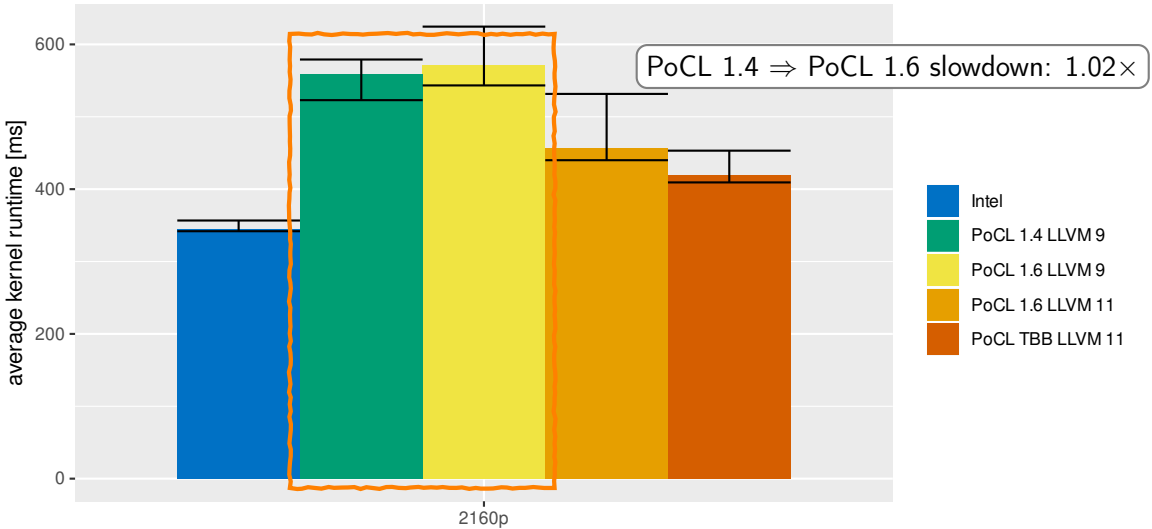
Real World Application Benchmark: raytracing



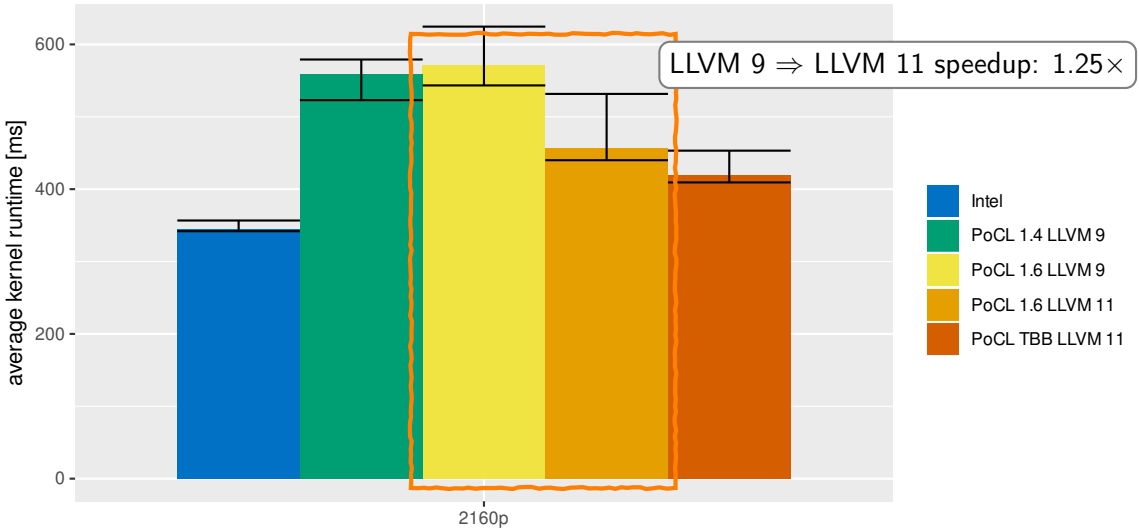
Real World Application Benchmark: raytracing



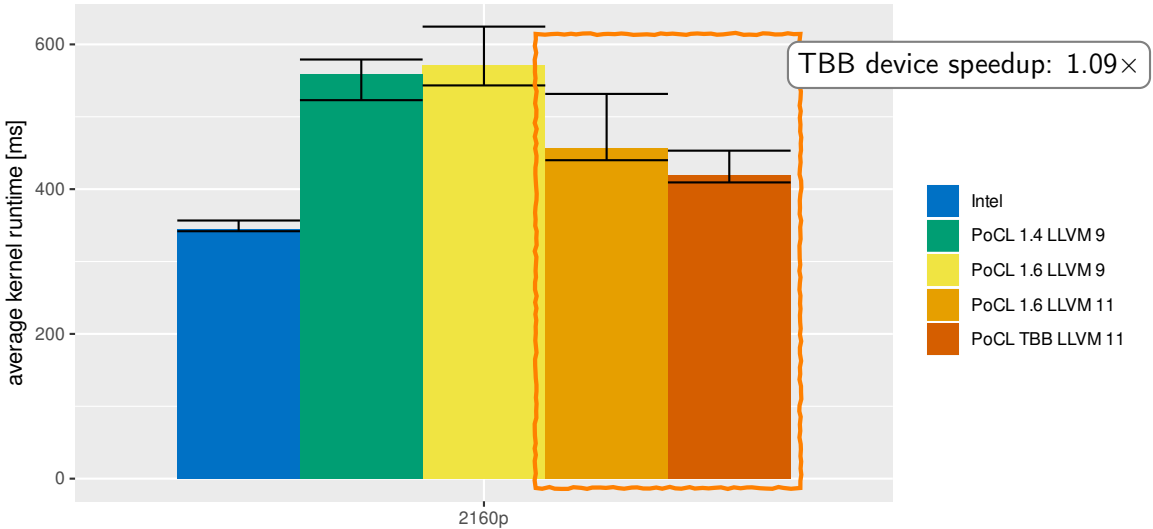
Real World Application Benchmark: raytracing



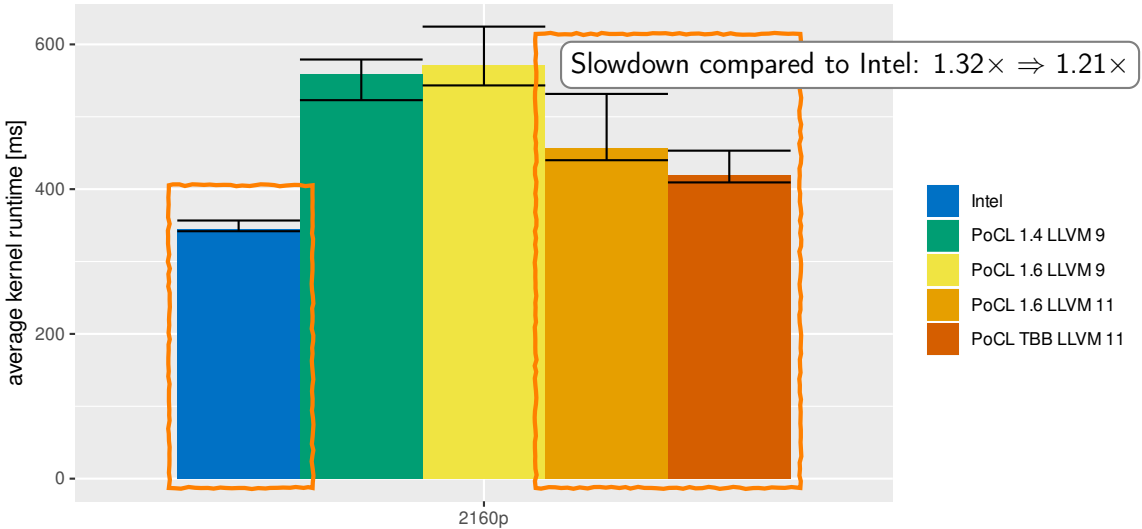
Real World Application Benchmark: raytracing



Real World Application Benchmark: raytracing



Real World Application Benchmark: raytracing



Vectorisation in PoCL

manual vectorisation

automatic vectorisation

Vectorisation in PoCL

manual vectorisation

```
void op_kernel (double8* in_a,
                double8* in_b,
                double8* out)
{
  // ...
  for (/* ITERATIONS */)
  {
    tmp_a += tmp_a * tmp_b;
  }
  // ...
}
```

automatic vectorisation

Vectorisation in PoCL

manual vectorisation

```
void op_kernel (double8* in_a,
                double8* in_b,
                double8* out)
{
  // ...
  for (/* ITERATIONS */)
  {
    tmp_a += tmp_a * tmp_b;
  }
  // ...
}
```

vfmadd231**pd** %zmm1,%zmm2,%zmm3

⇒ **trivial** for the compiler

⇒ 8 doubles per OpenCL work-item

automatic vectorisation

Vectorisation in PoCL

manual vectorisation

```
void op_kernel (double8* in_a,
                double8* in_b,
                double8* out)
{
  // ...
  for (/* ITERATIONS */)
  {
    tmp_a += tmp_a * tmp_b;
  }
  // ...
}
```

automatic vectorisation

```
void op_kernel (double* in_a,
                double* in_b,
                double* out)
{
  // ...
  for (/* ITERATIONS */)
  {
    tmp_a += tmp_a * tmp_b;
  }
  // ...
}
```

vfmadd231**pd** %zmm1,%zmm2,%zmm3

⇒ **trivial** for the compiler

⇒ 8 doubles per OpenCL work-item

Vectorisation in PoCL

manual vectorisation

```
void op_kernel (double8* in_a,
                double8* in_b,
                double8* out)
{
  // ...
  for (/* ITERATIONS */)
  {
    tmp_a += tmp_a * tmp_b;
  }
  // ...
}
```

vfmadd231**pd** %zmm1,%zmm2,%zmm3

⇒ **trivial** for the compiler

⇒ 8 doubles per OpenCL work-item

automatic vectorisation

```
void op_kernel (double* in_a,
                double* in_b,
                double* out)
{
  // ...
  for (/* ITERATIONS */)
  {
    tmp_a += tmp_a * tmp_b;
  }
  // ...
}
```

vfmadd231**sd** %xmm0,%xmm1,%xmm0

⇒ **no vectorisation** across work-items

⇒ inner loop not vectorisable (reduction)

Goal: Automatic Work-Item Vectorisation in PoCL

Goal: Automatic Work-Item Vectorisation in PoCL

- **SIMD** vectorisation across work-items on CPUs $\approx$ **SIMT** execution on GPUs
  - $\Rightarrow$ requires **outer-loop vectorisation** outside kernel function
  - $\Rightarrow$ implemented in Intel OpenCL, but **not in PoCL**

Goal: Automatic Work-Item Vectorisation in PoCL

- **SIMD** vectorisation across work-items on CPUs $\approx$ **SIMT** execution on GPUs
  - $\Rightarrow$ requires **outer-loop vectorisation** outside kernel function
  - $\Rightarrow$ implemented in Intel OpenCL, but **not in PoCL**
- **Feasibility-evaluation** through simplified OpenMP-proxy of OpenCL runtime:

Goal: Automatic Work-Item Vectorisation in PoCL

- **SIMD** vectorisation across work-items on CPUs ≈ **SIMT** execution on GPUs
  - ⇒ requires **outer-loop vectorisation** outside kernel function
  - ⇒ implemented in Intel OpenCL, but **not in PoCL**
- **Feasibility-evaluation** through simplified OpenMP-proxy of OpenCL runtime:

```
/* automatic vectorisation scheme */
#pragma omp parallel for // threading across WGs
for (int group_id = 0; group_id < (num / VEC_LENGTH); ++group_id)
{
    /* vectorised work-item loop */
    #pragma omp simd // vectorisation across WIs inside WG
    for (int local_id = 0; local_id < VEC_LENGTH; ++local_id)
    {
        // scalar typed kernel code, e.g. hexciton_benchmark
    }
}
```

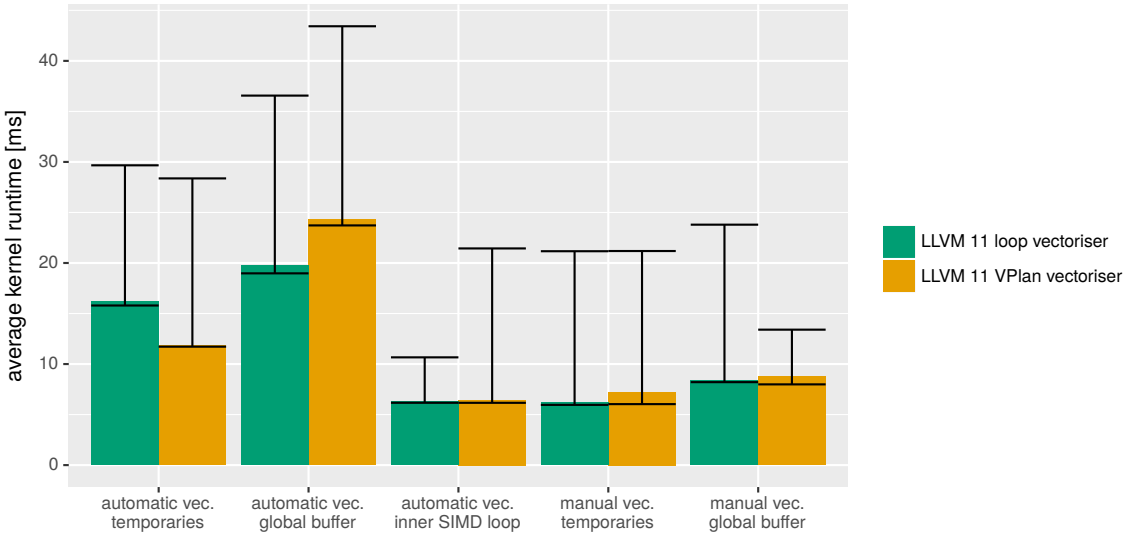
Goal: Automatic Work-Item Vectorisation in PoCL

- **SIMD** vectorisation across work-items on CPUs ≈ **SIMT** execution on GPUs
  - ⇒ requires **outer-loop vectorisation** outside kernel function
  - ⇒ implemented in Intel OpenCL, but **not in PoCL**
- **Feasibility-evaluation** through simplified OpenMP-proxy of OpenCL runtime:

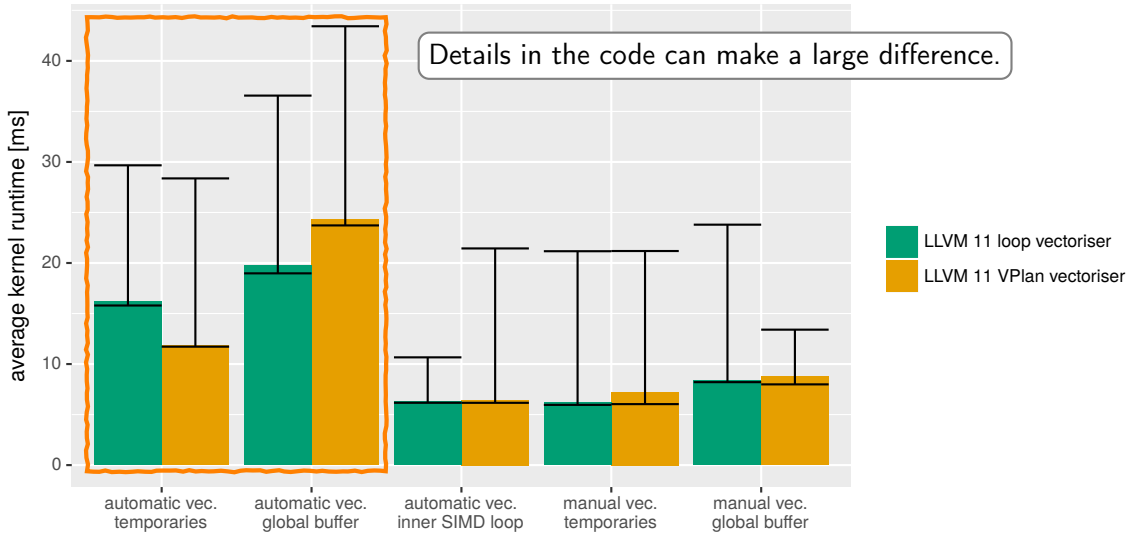
```
/* automatic vectorisation scheme */
#pragma omp parallel for // threading across WGs
for (int group_id = 0; group_id < (num / VEC_LENGTH); ++group_id)
{
    /* vectorised work-item loop */
    #pragma omp simd // vectorisation across WIs inside WG
    for (int local_id = 0; local_id < VEC_LENGTH; ++local_id)
    {
        // scalar typed kernel code, e.g. hexciton_benchmark
    }
}
```

Can LLVM 11 do this?

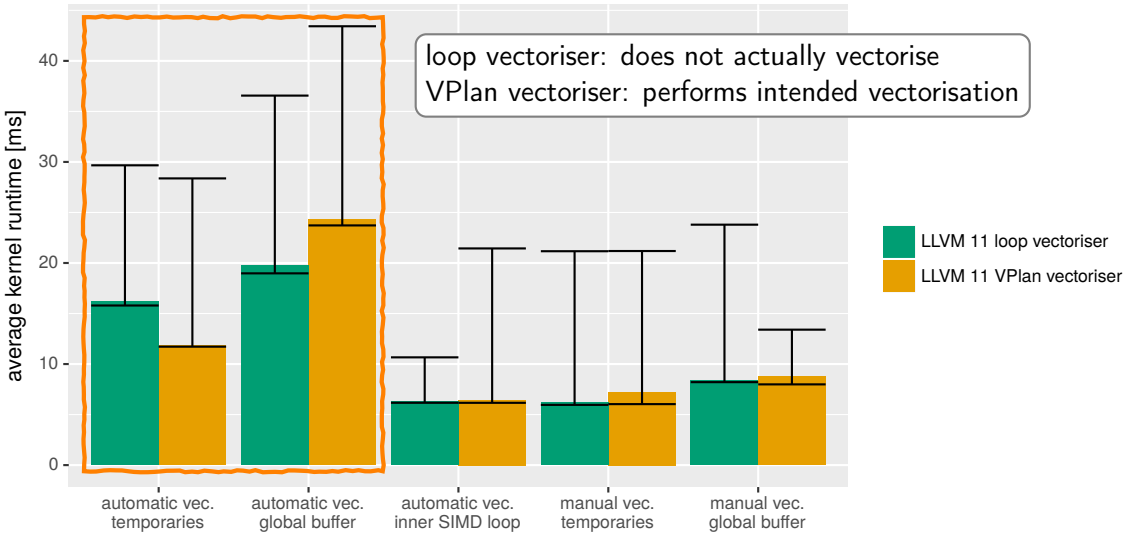
hexciton_benchmark: LLVM Vectoriser Comparison



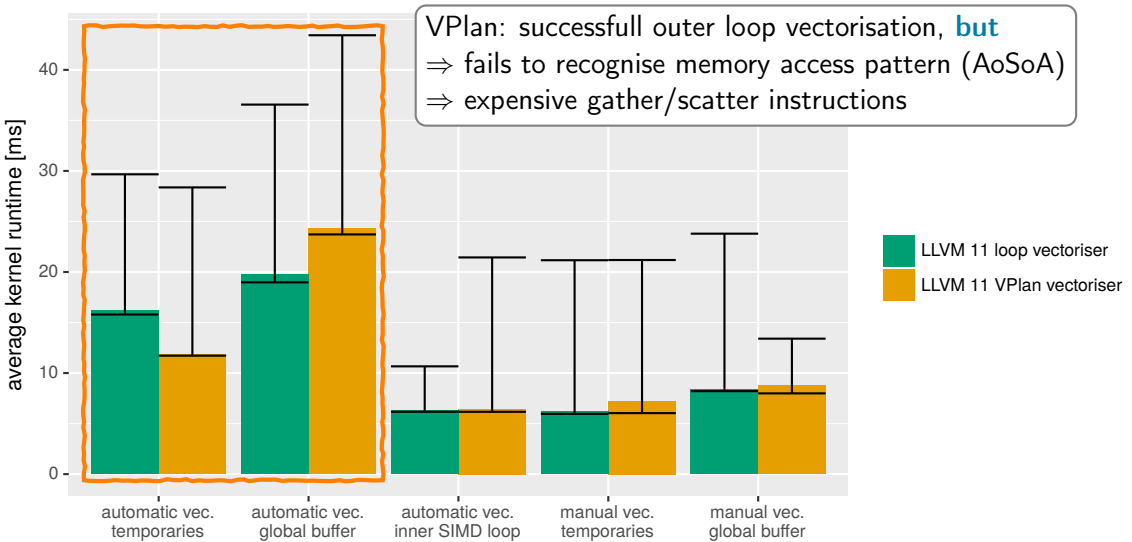
hexciton_benchmark: LLVM Vectoriser Comparison



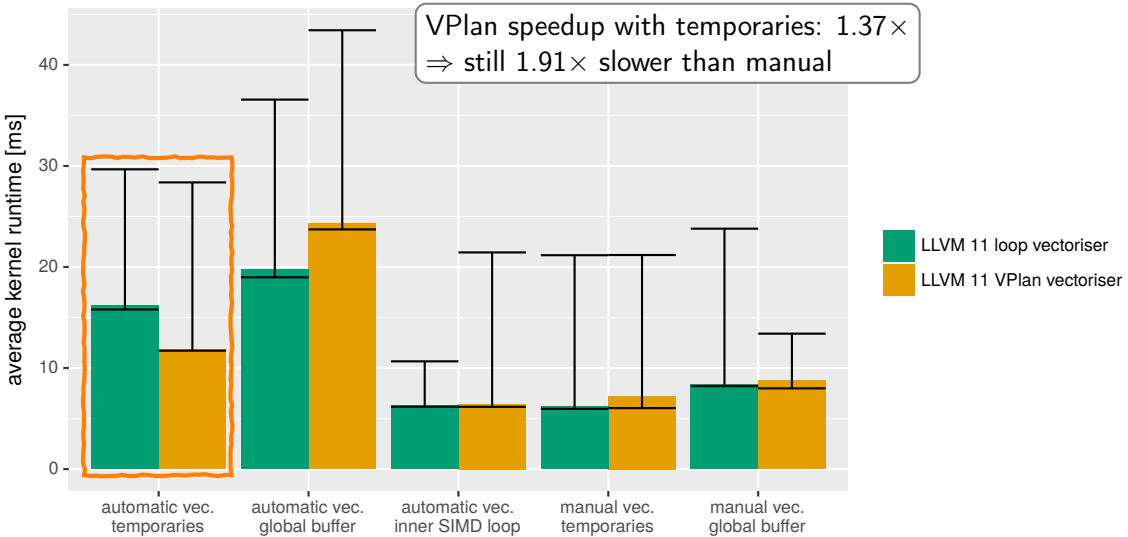
hexciton_benchmark: LLVM Vectoriser Comparison



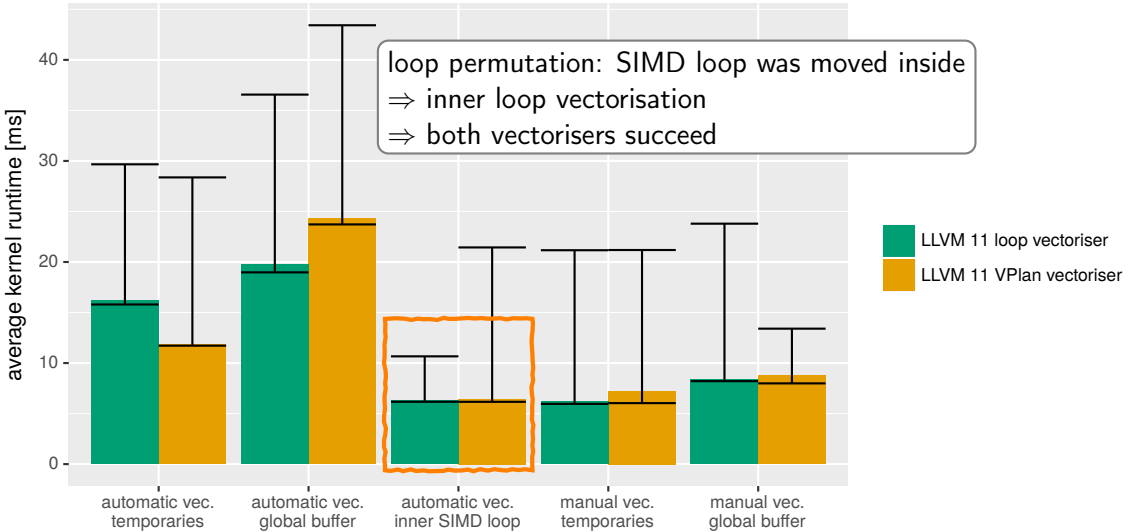
hexciton_benchmark: LLVM Vectoriser Comparison



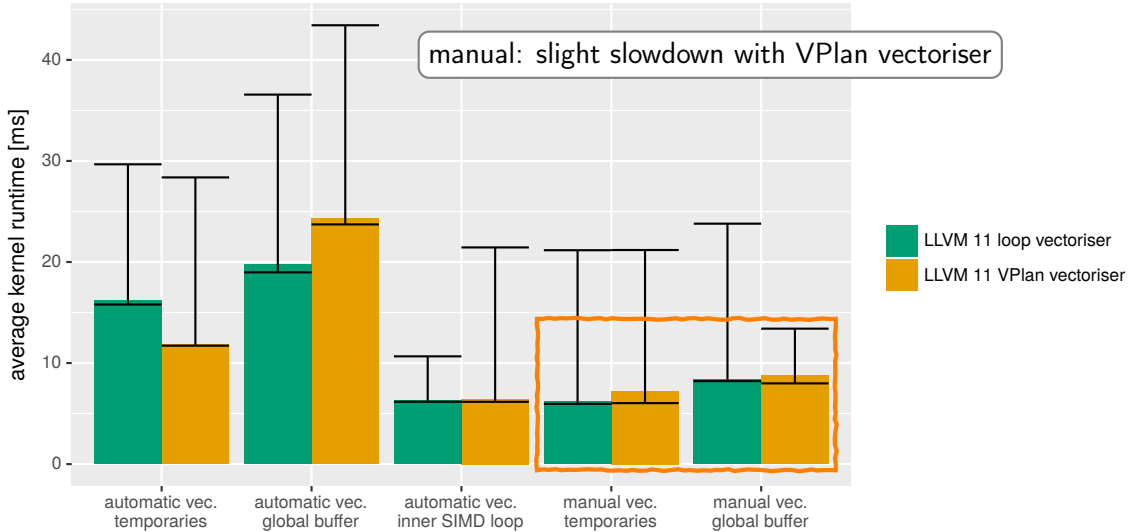
hexciton_benchmark: LLVM Vectoriser Comparison



hexciton_benchmark: LLVM Vectoriser Comparison



hexciton_benchmark: LLVM Vectoriser Comparison



Summary

- Performance Evaluation PoCL vs. Intel OpenCL on Intel CPUs
  - comparable performance only with manual vectorisation

Tobias Baumann, Matthias Noack, Thomas Steinke. 2021. Performance Evaluation and Improvements of the *PoCL* Open-Source OpenCL Implementation on Intel CPUs. *International Workshop on OpenCL (IWOCL'21)*. https://doi.org/10.1145/3456669.3456698

Funded by:
German Federal Ministry of Education and Research (BMBF), Research Campus MODAL, project no. 05M20ZBM

Summary

- Performance Evaluation PoCL vs. Intel OpenCL on Intel CPUs
  - comparable performance only with manual vectorisation
- TBB-based device for PoCL
  - speedups over Intel possible (achieved: 1.3×)
- Two bugs fixed in PoCL

Tobias Baumann, Matthias Noack, Thomas Steinke. 2021. Performance Evaluation and Improvements of the *PoCL* Open-Source OpenCL Implementation on Intel CPUs. *International Workshop on OpenCL (IWOCL'21)*. https://doi.org/10.1145/3456669.3456698

Funded by:
German Federal Ministry of Education and Research (BMBF), Research Campus MODAL, project no. 05M20ZBM

Summary

- Performance Evaluation PoCL vs. Intel OpenCL on Intel CPUs
  - comparable performance only with manual vectorisation
- TBB-based device for PoCL
  - speedups over Intel possible (achieved: 1.3×)
- Two bugs fixed in PoCL
- LLVM vectoriser evaluation
  - outer loop vectorisation across work-items in PoCL depends on LLVM's VPlan progress

Tobias Baumann, Matthias Noack, Thomas Steinke. 2021. Performance Evaluation and Improvements of the *PoCL* Open-Source OpenCL Implementation on Intel CPUs. *International Workshop on OpenCL (IWOCL'21)*. https://doi.org/10.1145/3456669.3456698

Funded by:
German Federal Ministry of Education and Research (BMBF), Research Campus MODAL, project no. 05M20ZBM

Summary

- Performance Evaluation PoCL vs. Intel OpenCL on Intel CPUs
  - comparable performance only with manual vectorisation
- TBB-based device for PoCL
  - speedups over Intel possible (achieved: 1.3×)
- Two bugs fixed in PoCL
- LLVM vectoriser evaluation
  - outer loop vectorisation across work-items in PoCL depends on LLVM's VPlan progress

Thank you for your attention!

tobias.baumann@zib.de

Tobias Baumann, Matthias Noack, Thomas Steinke. 2021. Performance Evaluation and Improvements of the *PoCL* Open-Source OpenCL Implementation on Intel CPUs. *International Workshop on OpenCL (IWOCL'21)*. https://doi.org/10.1145/3456669.3456698

Funded by:
German Federal Ministry of Education and Research (BMBF), Research Campus MODAL, project no. 05M20ZBM