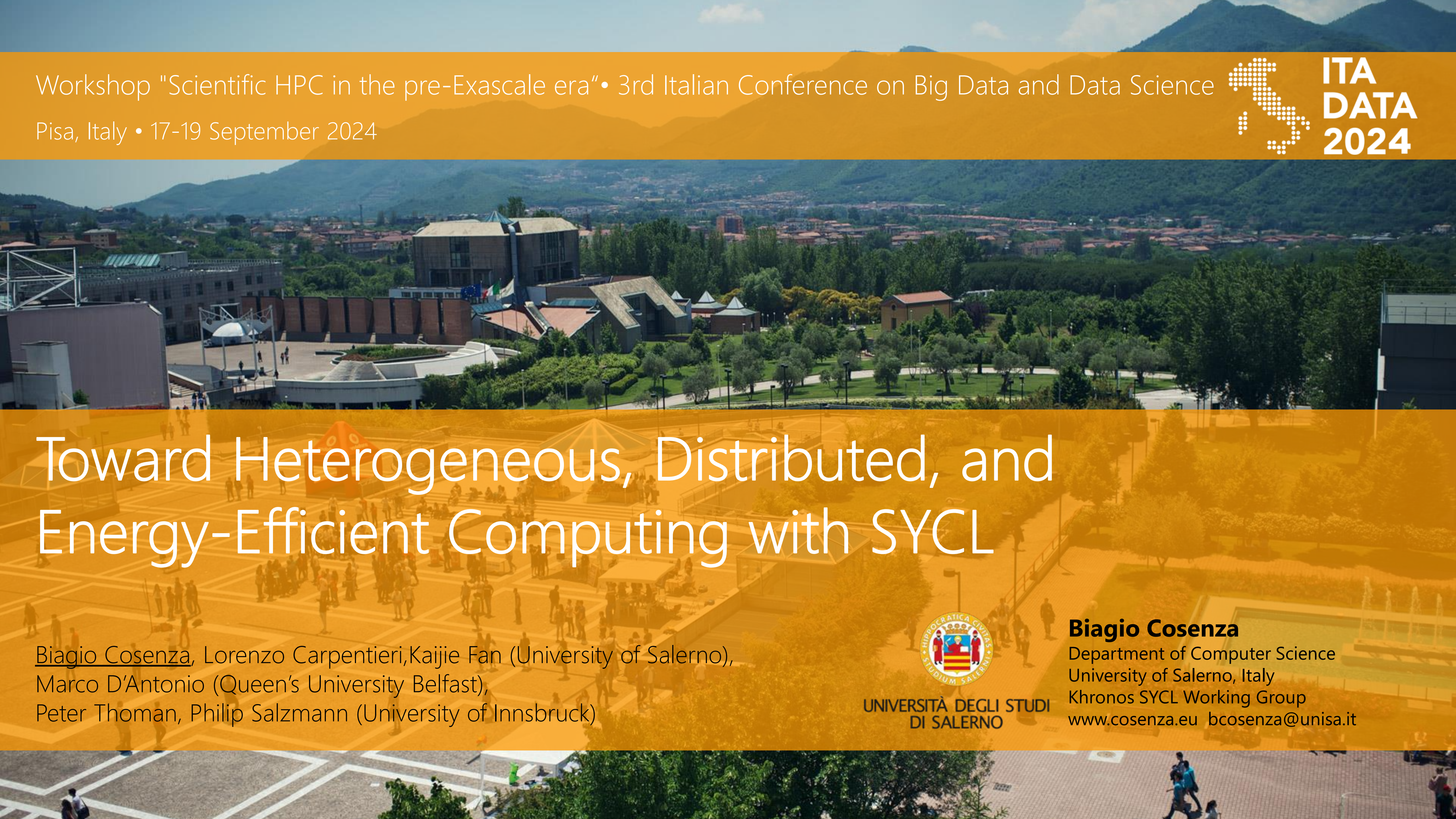


Workshop "Scientific HPC in the pre-Exascale era"• 3rd Italian Conference on Big Data and Data Science

Pisa, Italy • 17-19 September 2024

ITA DATA 2024

Toward Heterogeneous, Distributed, and Energy-Efficient Computing with SYCL

Biagio Cosenza, Lorenzo Carpentieri,Kaijie Fan (University of Salerno),
Marco D'Antonio (Queen's University Belfast),
Peter Thoman, Philip Salzmann (University of Innsbruck)

UNIVERSITÀ DEGLI STUDI DI SALERNO

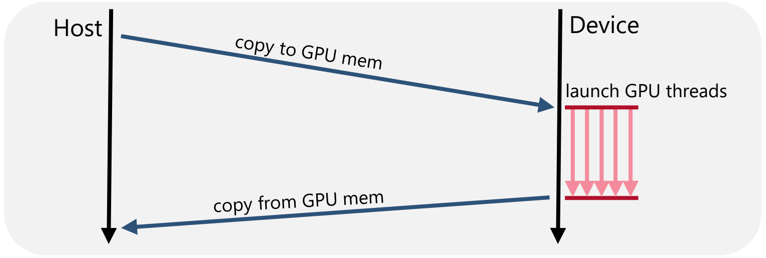
Biagio Cosenza
Department of Computer Science
University of Salerno, Italy
Khronos SYCL Working Group
www.cosenza.eu bcosenza@unisa.it

Outline

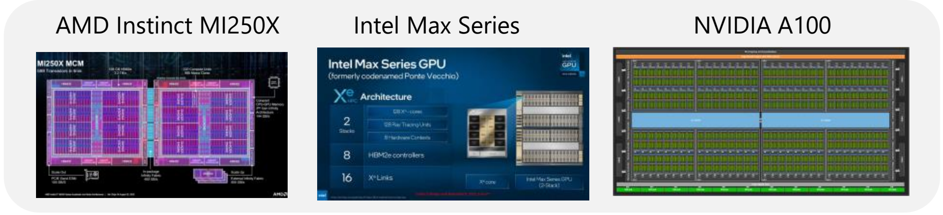
- Programming Model Challenges
- Heterogenous programming with SYCL
- Extending SYCL semantics to provide additional features
    - Programming cluster of accelerator (Celerity)
    - Energy efficient computing (SYnergy)
    - WIP: Approximate computing (SYprox)

Programming the Exascale: Challenges

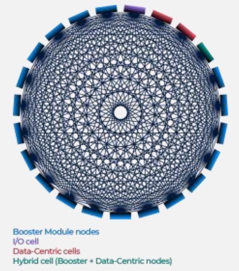
1. Heterogenous Programming
   - Challenges: data handling, GPU opt., massive parallelism



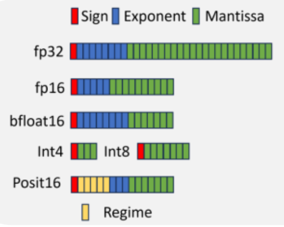
2. Portability and performance portability
   - Challenges: different vendor progr. models, different tuning

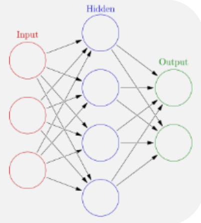


3. Distributed programming: cluster of GPUs
   - Challenges: partitioning, data movement, scheduling



4. Optimizing for emerging workloads (AI)
   - Challenges: mixed precision, other approximate computing techniques



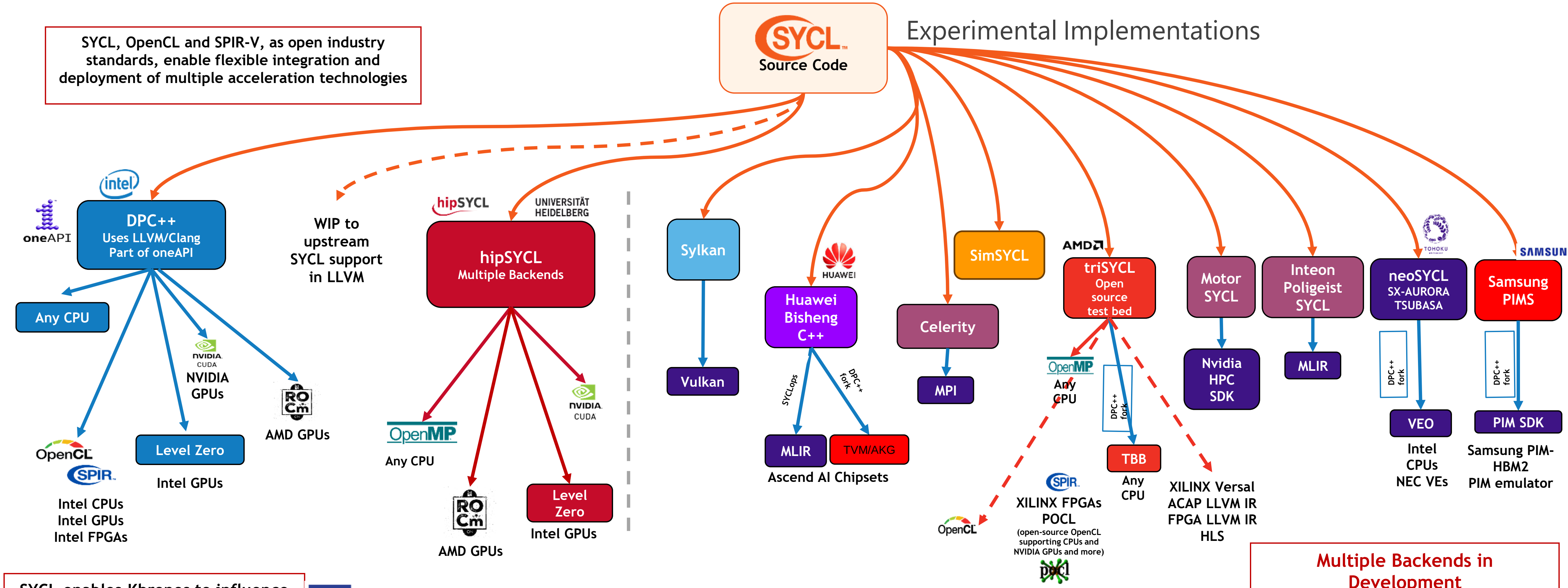


Introduction to SYCL

- SYCL is a single source, high-level, standard C++ programming model, that can target a range of heterogeneous platforms
- Open standard by Khronos Group
  - SYCL and the SYCL logo are trademarks of the Khronos Group Inc.
  - University of Salerno is Khronos Group Member
- Enables programming for heterogenous hardware from different vendors
  - CPU, GPU, FPGA, accelerators
  - freedom from vendor lock-in
- Comparable performance to native CUDA
  - Migration tool: SYCLomatic

SYCL Implementations

SYCL, OpenCL and SPIR-V, as open industry standards, enable flexible integration and deployment of multiple acceleration technologies



SYCL enables Khronos to influence ISO C++ to (eventually) support heterogeneous compute

Multiple Backends in Development

SYCL on even more low-level frameworks.
For more information: http://sycl.tech

Edited image, original image courtesy of Michael Wong, CodePlay Ltd

SYCL Code Example

- Single source using a sycl header
- A queue point to the device
  - GPU device
- Buffer and accessors
  - Buffers handle data and synchronize on destruction via RAII
  - Accessors specify read/write access to a buffer

```
#include <iostream>
#include <sycl/sycl.hpp>
using namespace sycl;

int main(int, char**) {
  const int size = 10000;
  std::vector<float> x_vec(size, 1.0f);
  std::vector<float> y_vec(size, 2.0f);
  float a = 0.5;

  queue q(gpu_selector_v);
  buffer x_buf(x_vec);
  buffer y_buf(y_vec);
  range<1> num_items{ x_vec.size() };
  q.submit([&](handler& h) {
    accessor x(x_buf, h, read_only);
    accessor y(y_buf, h, read_write);
    h.parallel_for(num_items, [=](item<1> i) {
      y[i] = a * x[i] + y[i];
    });
  });
  host_accessor y_res(y_buf, read_only);
  // … print results and returns
}
```

SYCL Code Example

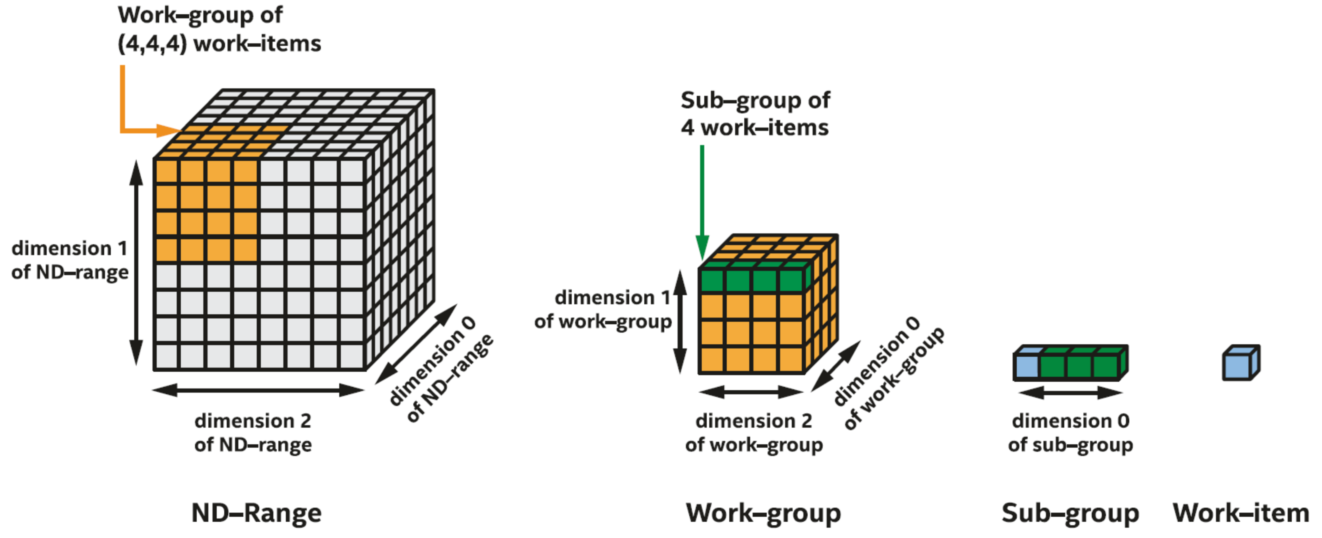
- Parallelism expressed by řǻsǻľľêľ ǧộs
  - lamba function executed on the GPU
- Final synchronization by host accessor
  - access to data in a buffer from host
  - blocking call
  - return after operation complete

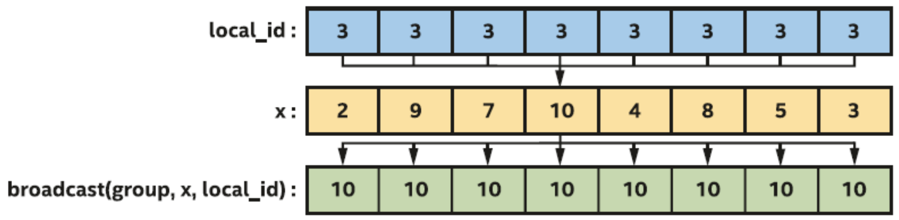
```
#include <iostream>
#include <sycl/sycl.hpp>
using namespace sycl;

int main(int, char**) {
  const int size = 10000;
  std::vector<float> x_vec(size, 1.0f);
  std::vector<float> y_vec(size, 2.0f);
  float a = 0.5;

  queue q(gpu_selector_v);
  buffer x_buf(x_vec);
  buffer y_buf(y_vec);
  range<1> num_items{ x_vec.size() };
  q.submit([&](handler& h) {
    accessor x(x_buf, h, read_only);
    accessor y(y_buf, h, read_write);
    h.parallel_for(num_items, [=](item<1> i) {
      y[i] = a * x[i] + y[i];
    });
  });
  host_accessor y_res(y_buf, read_only);
  // … print results and returns
}
```

Image courtesy from *Data Parallel C++ Mastering DPC++ for Programming of Heterogeneous Systems using C++ and SYCL*

SYCL Advanced Features

- Unified Shared Memory
- Work-group
- Sub-groups and group algorithms
- Local memory
- Atomic operations
- Kernel reductions
- Specialization constants


Work–group of (4,4,4) work–items
dimension 1 of ND–range
dimension 2 of ND–range
dimension 0 of ND–range
ND–Range
Sub–group of 4 work–items
dimension 1 of work–group
dimension 2 of work–group
dimension 0 of work–group
Work–group
dimension 0 of sub–group
Sub–group
Work–item



```
queue q;
float* x = malloc_shared<float>(size, q);
float* y = malloc_shared<float>(size, q);
// ...
range<1> num_items{ size };
q.submit([&](handler& h) {
  h.parallel_for(num_items, [=](item<1> i) {
    y[i] = a * x[i] + y[i];
  });
});
q.wait();
// ... print results
free(x, q);
free(y, q);
```

Performance portability study on industrial application

Crisci, Carpentieri, Cosenza, Accordi, Gadioli, Vitali, Palermo, Beccari: Enabling performance portability on the LiGen drug discovery pipeline. Future Gener. Comput. Syst. 158: 44-59 (2024)

SYCL Extensions Overview

Celerity

Aims: Programming cluster of accelerators

Celerity extends SYCL queue with a `distr_queue` and accessor semantics with range mappers that express data access patterns

```
distr_queue q(gpu_selector);
q.submit([&] (handler& h) {
accessor x{x_buf,h,read_only, one_to_one};
accessor y{y_buf,h,read_only, one_to_one};
accessor z{z_buf,h,write_only,one_to_one};
  h.parallel_for(range<1>(N),
    [=](id<1> i){
    z[i] = alpha * x[i] + y[i];
  });
});
```

SYnergy

Aims: Energy-efficient computing

SYnergy proposes an energy-aware queue that allow for energy measurement, per-queue and per-kernel frequency scaling, and support for different energy target

```
synergy::queue q (gpu_selector, MIN_EDP);
q.submit([&] (handler& h) {
  accessor x {x_buf,h,read_only};
  accessor y {y_buf,h,read_only};
  accessor z {z_buf,h,write_only};
  h.parallel_for(range<1>(N),
    [=](id<1> i){
    z[i] = alpha * x[i] + y[i];
  });
});
```

SYprox

Aims: Approximate computing

SYprox implements advanced approximate computing techniques such as kernel perforation with signal reconstruction and mixed precision, e.g., with perforated `paccessor`

```
queue q(gpu_selector);
q.submit([&] (handler& h) {
  paccessor x {x_buf,h,read_only};
  paccessor y {y_buf,h,read_only};
  accessor  z {z_buf,h,write_only};
  h.parallel_for(range<1>(N),
    [=](id<1> i){
      z[i] = alpha * x[i] + y[i];
    });
});
```

Celerity

Celerity
High-level C++ for Accelerator Clusters
https://github.com/celerity

- High-level API designed from the ground up for accelerator clusters
  - allows to constrain data structures and processing patterns to ones efficient on accelerators less complex than fully general distributed memory programming
- Based on SYCL
- Aim to provide easy porting path for SYCL applications
  - no explicit distribution, synchronization or communication
  - derived entirely from data flow
- Carried out at the University of Innsbruck (lead) and the University of Salerno

Peter Thoman, Philip Salzmann, Biagio Cosenza, Thomas Fahringer: Celerity: High-Level C++ for Accelerator Clusters. Euro-Par 2019: 291-303

Celerity: from SYCL to Celerity

Celerity
High-level C++ for Accelerator Clusters
https://github.com/celerity

- SYCL-based interface
  - minimum divergence from SYCL
  - allow to easily target cluster of GPUs
  - new concepts: distributed queue, range mappers
- Distributed runtime system
  - multi-pass execution
  - automatic, implicit data movement between devices
  - task and command graph generation
  - fully asynchronous data and command exchange

A SAXPY kernel in SYCL

```
queue q(gpu_selector);
q.submit([&] (handler& h) {
accessor x{x_buf,h,read_only};
accessor y{y_buf,h,read_only};
accessor z{z_buf,h,write_only};
  h.parallel_for(range<1>(N),
    [=](id<1> i){
    z[i] = alpha * x[i] + y[i];
  });
});
```

A SAXPY kernel in Celerity

```
distr_queue q(gpu_selector);
q.submit([&] (handler& h) {
accessor x{x_buf,h,read_only, one_to_one};
accessor y{y_buf,h,read_only, one_to_one};
accessor z{z_buf,h,write_only, one_to_one};
  h.parallel_for(range<1>(N),
    [=](id<1> i){
    z[i] = alpha * x[i] + y[i];
  });
});
```

Celerity Range Mappers

Celerity
High-level C++ for Accelerator Clusters
https://github.com/celerity

- Range mappers
  - arbitrary functors
  - mapping from a K-dimensional kernel index space chunk to a B-dimensional buffer index space subrange
- Example of predefined range mappers
  - `One-to-one`
  - `Neighbourhood<2>`
  - `Slice<2>`

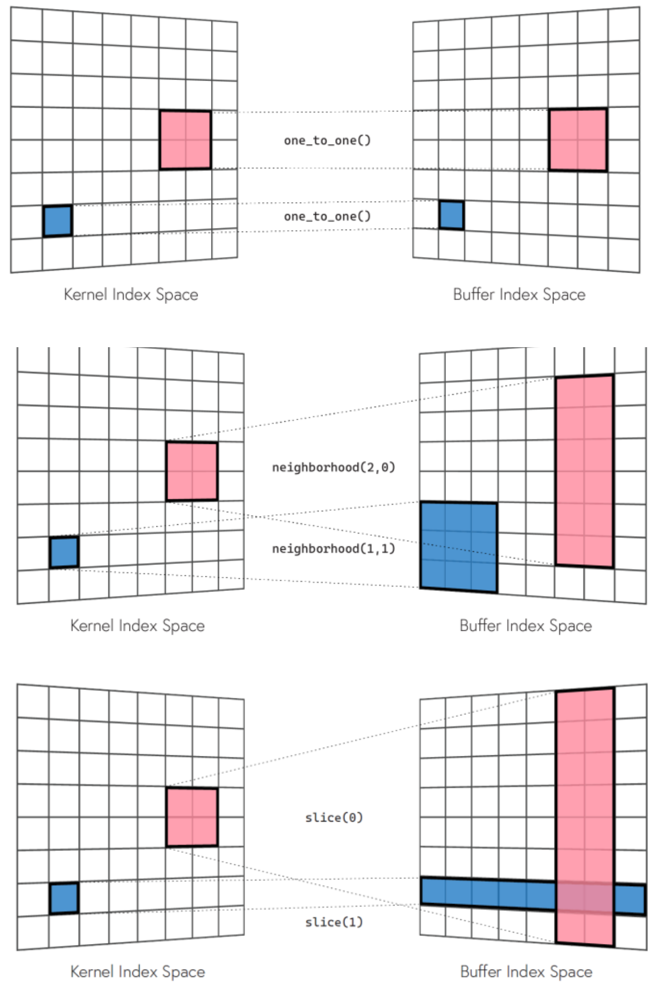
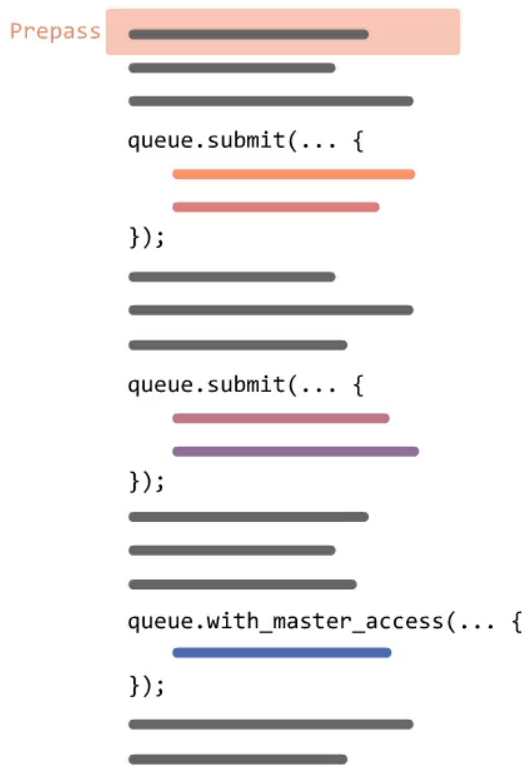


Image courtesy of Peter Thoman, University of Innsbruck

Celerity Distributed Runtime Systems

https://github.com/celerity

Program Code **Task Graph** **Command Graph**



```
queue.submit(... {
});
queue.submit(... {
});
queue.with_master_access(... {
});
```

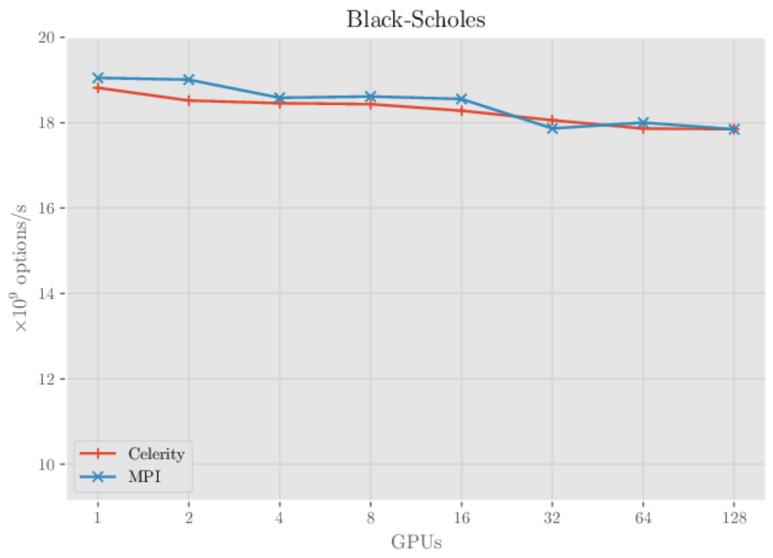
An Asynchronous Dataflow-Driven Execution Model For Distributed Accelerator Computing. Philip Salzmann, Fabian Knorr, Peter Thoman, Philipp Gschwandtner, Biagio Cosenza and Thomas Fahringer CCGrid 2023

LibWater: heterogeneous distributed computing made easy
Ivan Grasso, Simone Pellegrini, Biagio Cosenza, Thomas Fahringer. ICS 2013: 161-172

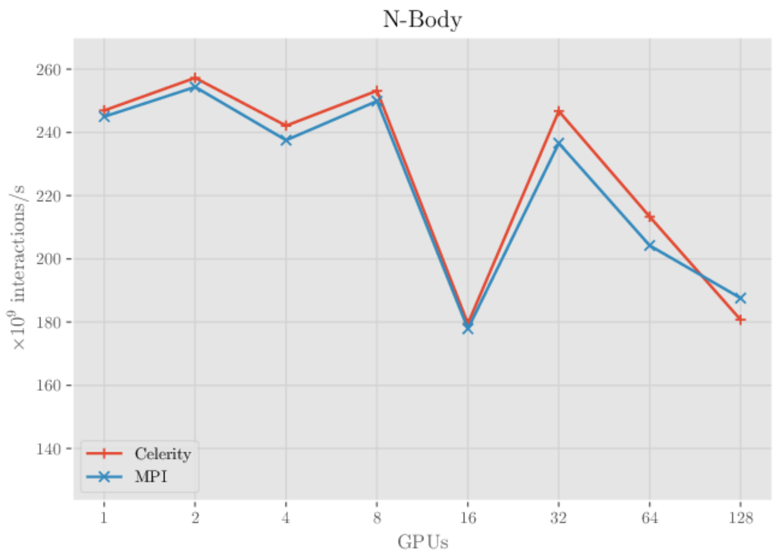
Celerity Scaling Results

Celerity
High-level C++ for Accelerator Clusters
https://github.com/celerity

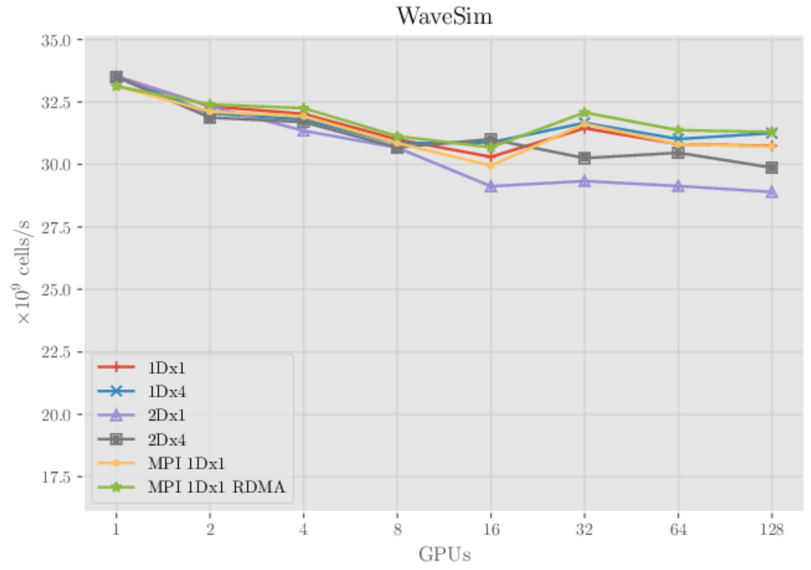
- Experiments on 128 GPUs (Marconi100 at CINECA)



>95% efficiency



>70% efficiency



>93% efficiency

Image courtesy of Peter Thoman, University of Innsbruck

Celerity Summary

Celerity
High-level C++ for Accelerator Clusters
https://github.com/celerity

- Celerity is a SYCL-based API and runtime system for distributed accelerator computing
- Good scalability: up to 128 GPUs on Marconi100 and Leonardo at CINECA
  - enabled by new distributed scheduling model and horizons concept
- Future research directions
  - leveraging collective communication
  - dynamic load/data distribution and graph scheduling
  - integration with other SYCL extensions

UNIVERSITÀ DEGLI STUDI DI SALERNO

universität innsbruck

Reference papers

Peter Thoman, Philip Salzmann, Biagio Cosenza, Thomas Fahringer:
Celerity: High-Level C++ for Accelerator Clusters. Euro-Par 2019: 291-303

An Asynchronous Dataflow-Driven Execution Model For Distributed Accelerator Computing. Philip Salzmann, Fabian Knorr, Peter Thoman, Philipp Gschwandtner, Biagio Cosenza and Thomas Fahringer CCGrid 2023

DFG Deutsche Forschungsgemeinschaft   FWF Der Wissenschaftsfonds.   LIGATE   EuroHPC Joint Undertaking

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956137. This research has been partially funded by the FWF (I 3388) and DFG (CO 1544/1-1, project number 360291326) as part of the CELERITY project.

SYnergy

https://github.com/unisa-hpc/SYnergy

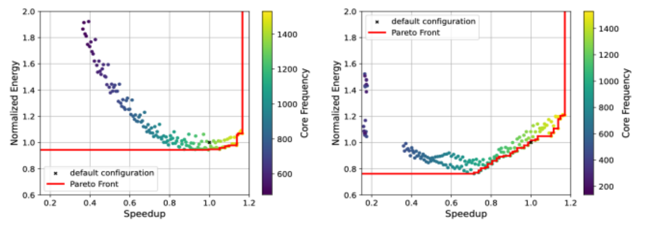
Synergy

- Dynamic Voltage and Frequency Scaling (DVFS) aims to reduce power consumption by dynamically adjusting voltage and frequency
  - Improve energy efficiency with DVFS on HPC systems
- Challenges
  1. No portable interface to support DVFS among different accelerators
  2. Frequency tuning must be done for each kernel
  3. Regular users do not have privilege to change frequency on HPC cluster

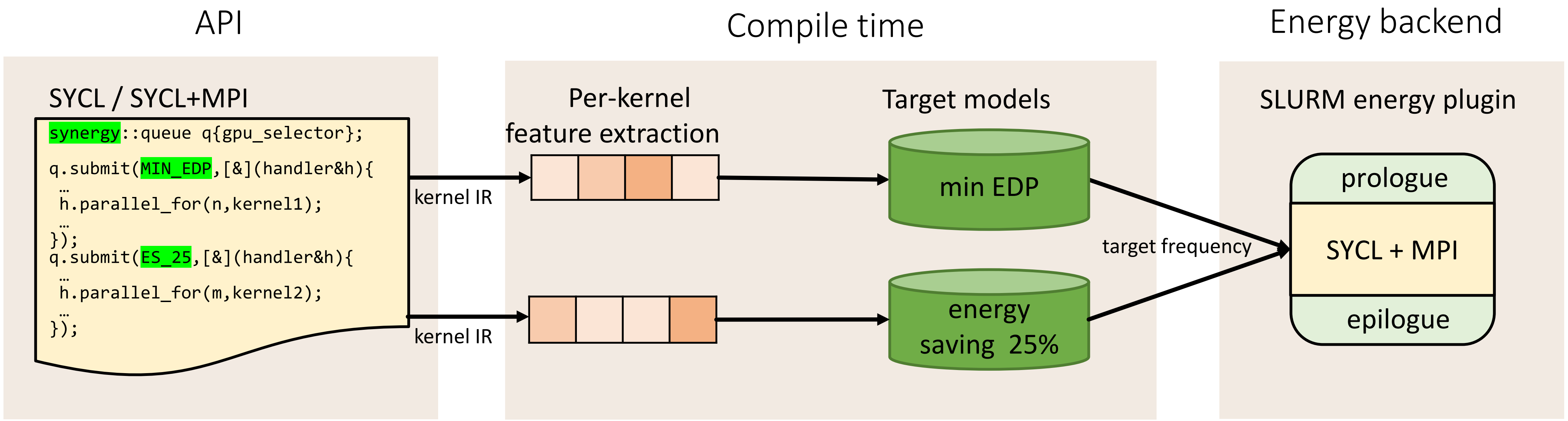
Level Zero

NVML

ROCm-SMI



SYnergy Overview



```
synergy::queue q{gpu_selector};
q.submit(MIN_EDP,[&](handler&h){
 …
 h.parallel_for(n,kernel1);
 …
});
q.submit(ES_25,[&](handler&h){
 …
 h.parallel_for(m,kernel2);
 …
});
```

- SYnergy interface: energy profiling; frequency scaling; energy target setting
  - synergy::queue and specify energy target metric at queue or kernel level
- SYnergy program compilation integrated with machine learning energy models
- Energy backend: native (LevelZero, NVML, ROcM); SLURM energy plugin to support GPU frequency scaling on cluster

SYnergy API: Energy Profiling

- Portable interface for providing energy profiling on heterogenous platforms
- Energy semantics
  - Energy-aware queue
  - Fine-grained energy profiling
  - Coarse-grained energy profiling

```
synergy::queue q{gpu_selector_v};
buffer<float, 1> x_buf{x};
buffer<float, 1> y_buf{y};

event e = q.submit([&](handler& h) {
  accessor<float, 1, read> x_acc{x_buf, h};
  accessor<float, 1, read> y_acc{y_buf, h};
  float a{alpha};

   h.parallel_for(range<1>{n},
     [=](id<1> id) {
    y_acc[id] = a * x_acc[id];
  });
});

double kernel_energy = q.kernel_energy_consumption(e);
double device_energy = q.device_energy_consumption();
}
```

SYnergy API: Frequency Scaling

- Portable interface for providing frequency scaling on heterogenous platforms
- Energy semantics
  - Energy-aware queue
  - Coarse-grained frequency scaling
  - Fine-grained frequency scaling

```
synergy::queue q1{1215, 210, gpu_selector_v};
synergy::queue q2{gpu_selector_v};

... // setup buffers

q1.submit([&](handler& h) {
   ... // setup accessors
   h.parallel_for(n,kernel1);
});

q2.submit(877,810, [&](handler& h) {
  ... // setup accessors
  h.parallel_for(m,kernel2);
});
```

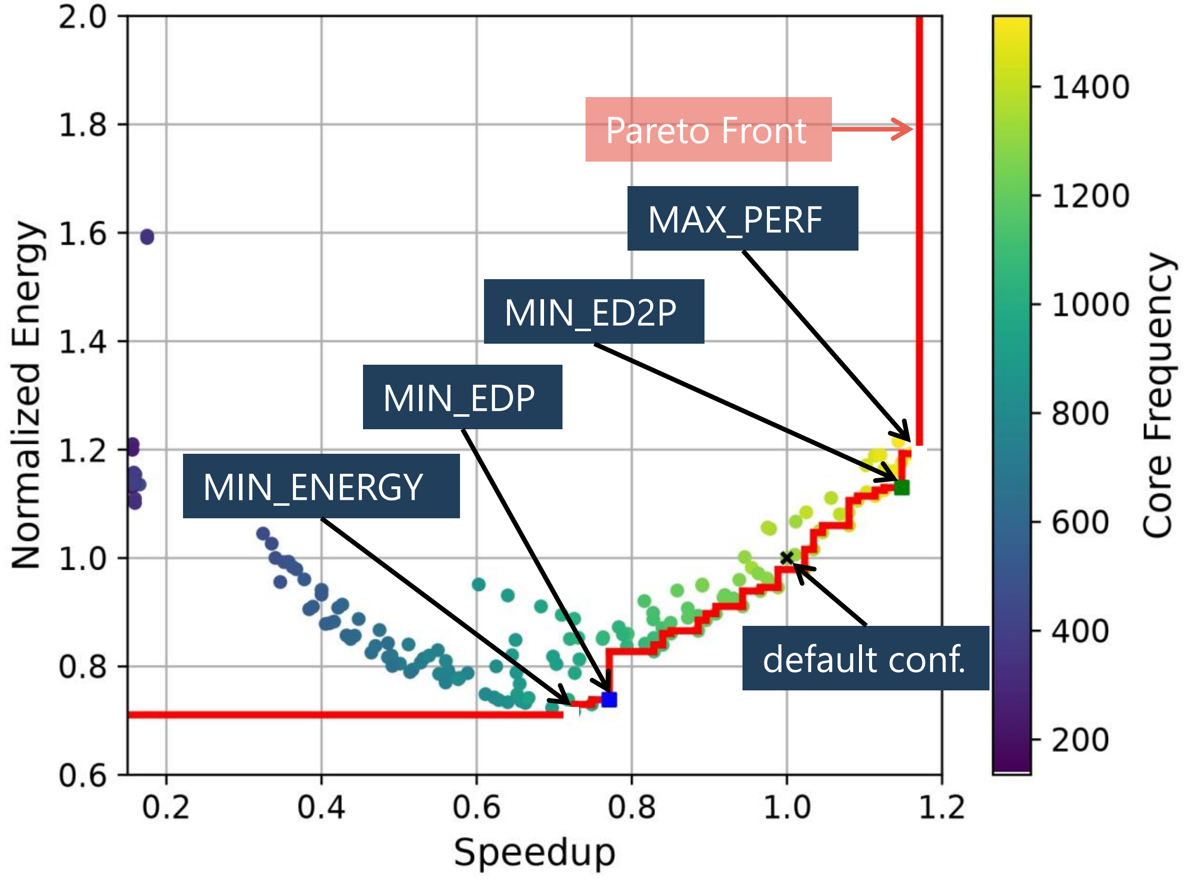
SYnergy API: Energy Targets

- Portable interface for selecting relevant frequency configurations
- Energy semantics
  - Energy-aware queue
  - Per-kernel energy target, e.g, MAX_PERF, MIN_ENERGY, MIN_EDP, or MIN_ED2P

```
synergy::queue q{gpu_selector_v};
buffer<float, 1> x_buf{x};
buffer<float, 1> y_buf{y};

event e = q.submit(MIN_EDP,[&](handler& h) {
    accessor<float, 1, read> x_acc{x_buf, h};
    accessor<float, 1, read> y_acc{y_buf, h};
    float a{alpha};

    h.parallel_for(range<1>{n}, [=](id<1> id) {
      y_acc[id] = a * x_acc[id];
    });
});
```

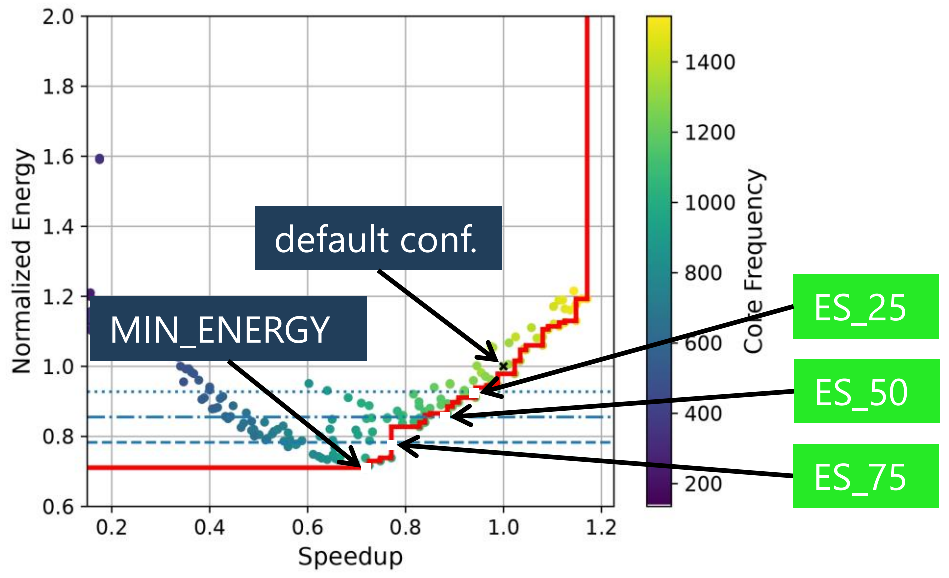
SYnergy API: Traditional Energy Targets

- Scalar metrics
  - `MAX_PERF`, `MIN_ENERGY`
  - `MIN_EDP`, `MIN_ED2P`
- Difficulty to represent energy-performance tradeoff
- Interesting configurations can be found in the multi-objective distribution

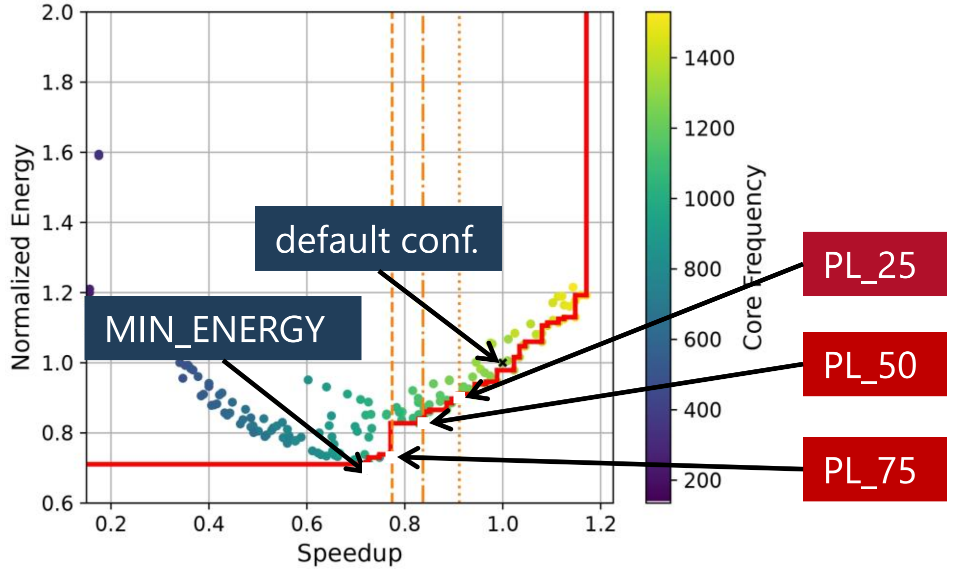


Energy targets of Black-Scholes benchmark

SYnergy API: Novel Energy Targets



Energy-saving targets



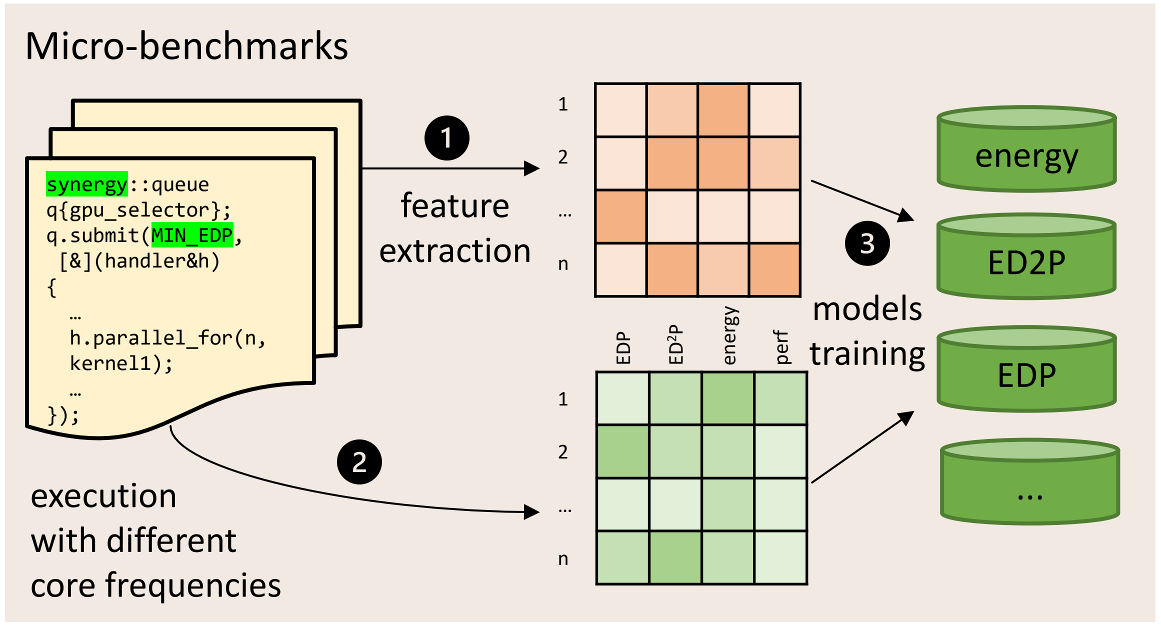
Performance-loss targets

New energy targets

- ES_x: the frequency configuration that delivers the x% relative energy savings
- PL_x: the frequency configuration that has x% relative performance loss
- Relative to the range [default, MIN_ENERGY]

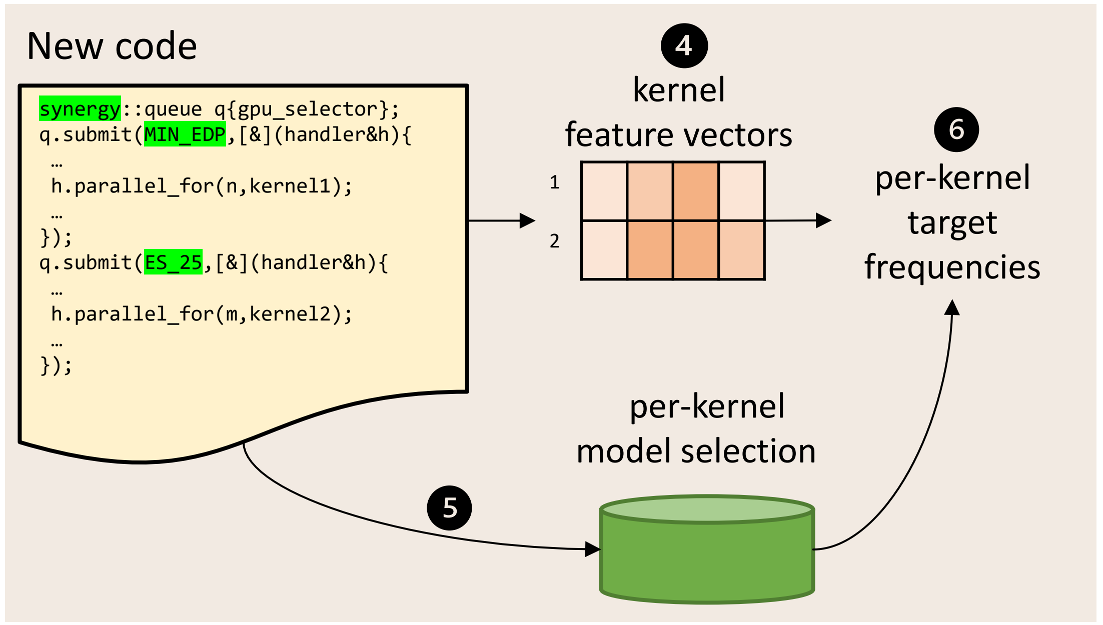
SYnergy Compilation: Energy Target Models

Training



```
synergy::queue
q{gpu_selector};
q.submit(MIN_EDP,
 [&](handler&h)
{
  …
  h.parallel_for(n,
  kernel1);
  …
});
```

Inference



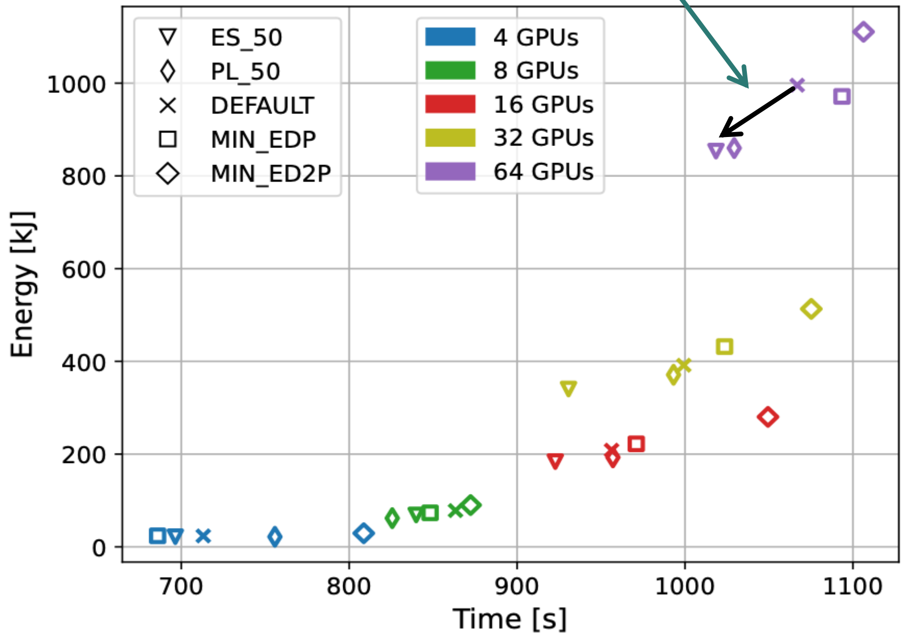
```
synergy::queue q{gpu_selector};
q.submit(MIN_EDP,[&](handler&h){
 …
 h.parallel_for(n,kernel1);
 …
});
q.submit(ES_25,[&](handler&h){
 …
 h.parallel_for(m,kernel2);
 …
});
```

- Energy characterization depends on the code & target hardware
  - kernel is characterized by static code features extracted by a LLVM pass
- Energy model based on machine learning
  - Training on microbenchmarks, evaluation on 23 benchmarks (SYCL-Bench), leave-one-out cross validation, each target use different models

Energy Scaling Evaluation on Marconi100 / CINECA

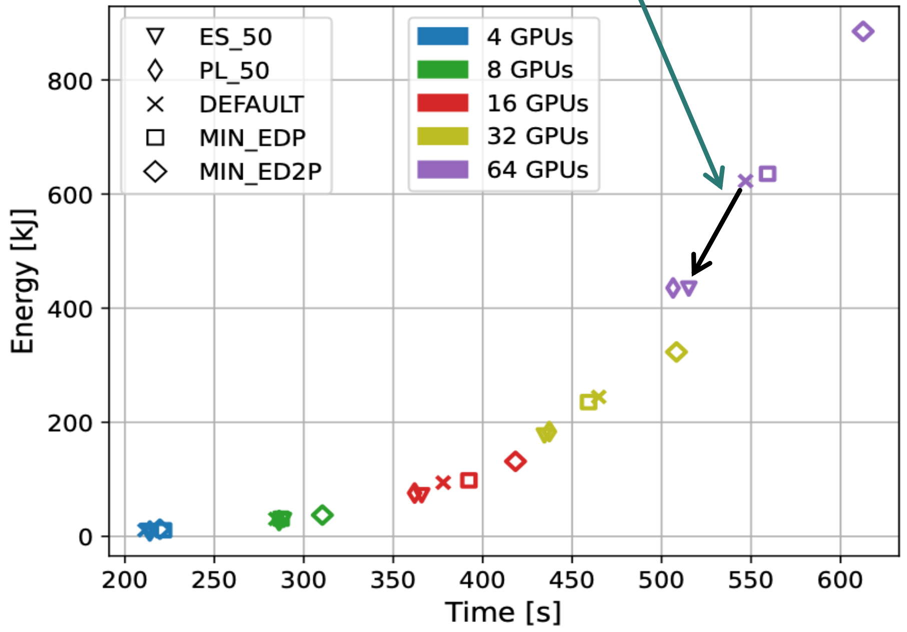
- Applications: CloverLeaf and MiniWeather

20% energy saving



(a) CloverLeaf

30% energy saving

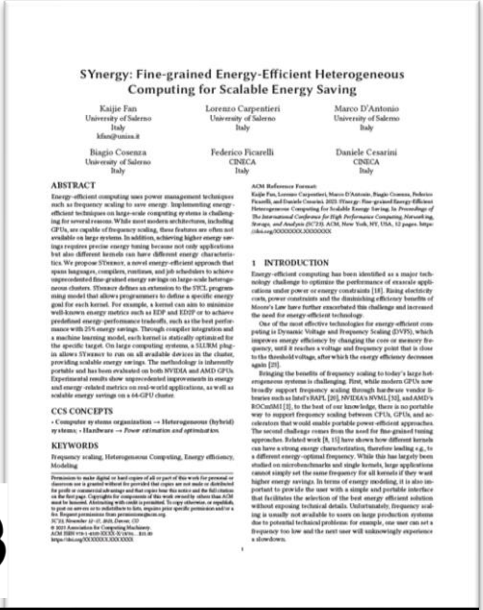


(b) MiniWeather

Summary: SYnergy

https://github.com/unisa-hpc/SYnergy

- SYnergy
  - SYCL interface for energy profiling and frequency scaling
  - Energy target and machine learning models
  - Energy backend for energy scalability on multiple GPUs
    - energy-aware SLURM plug-in
    - GEOPM, EAR
  - Support for domain-specific models


SYnergy: Fine-grained Energy-Efficient Heterogeneous Computing for Scalable Energy Saving

Kaijie Fan, University of Salerno, Italy, kfan@unisa.it
Lorenzo Carpentieri, University of Salerno, Italy
Marco D'Antonio, University of Salerno, Italy
Biagio Cosenza, University of Salerno, Italy
Federico Ficarelli, CINECA, Italy
Daniele Cesarini, CINECA, Italy

ABSTRACT

[illegible]

1 INTRODUCTION

[illegible]

SC23 DENVER NOV 12–17

Fan, Carpentieri, D'Antonio, Cosenza, Ficarelli, Cesarini: SYnergy: Fine-grained Energy-Efficient Heterogeneous Computing for Scalable Energy Saving. SC 2023

This project has received funding from the European High-Performance Computing Joint Undertaking Joint Undertaking (JU) under grant agreement No 956137. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Italy, Sweden, Austria, Czech Republic, Switzerland.

SYprox: SYCL for Approximate Computing

Work in progress

- Semantics to support a broad range of approximate computing techniques
- Perforation
  - Host perforation
  - Device perforation
  - with Perforation schemas
- Input and output reconstruction
- Data perforation + mixed precision

SYCL code with SYprox approximation

```
pbuffer<half,2,pcol<half> buf_a(a,range<2>{N,N});

// output reconstruction with lerp
pbuffer<half,2,pcol::lerp> out_buf(out,range<2>{N,N});

// global size and work group size
range<2> gl{N,N/2}, ws {32, 32};
q.submit([&](handler &h){
  paccessor<float,2,prow<float> > perf_acc{buf_a,h,read};
  accessor<float,2> acc_a{buf_a,h,read};
  h.parallel_for(nd_range<2>{gl,ws},
    [&](nd_item<2> it){
      id<2> id = it.get_global_id();
      // acc_a data are perforated host side
      out_acc[id*2] = acc_a[id] * 2;
      // perf_acc data are perforated device side
      out_acc[id*2] = perf_acc[id] * 2;
    });
}
```

Lorenzo Carpentieri, Biagio Cosenza: Towards a SYCL API for Approximate Computing. **IWOCL and SYCLcon 2023**:

Conclusions

- Heterogenous programming with SYCL
  - Basics: buffer/accessor and USM memory model, unorderd queue, …
  - Advanced: group algorithms, atomics, kernel reductions, …
- Extending SYCL semantics to provide additional features
  - Programming cluster of accelerator (Celerity)
  - Energy efficient computing (SYnergy)
  - Approximate computing (SYprox)

UNIVERSITÀ DEGLI STUDI DI SALERNO

Biagio Cosenza
Department of Computer Science
University of Salerno, Italy
Khronos SYCL Working Group
www.cosenza.eu bcosenza@unisa.it

Thanks to Lorenzo Carpentieri, Kaijie Fan, Luigi Crisci, Antonio De Caro, Majid Salimi Beni, Saleh Jamali Golzar (University of Salerno), Marco D'Antonio (Queen's University Belfast), Philip Salzmann, Fabian Knorr, Peter Thoman, Philipp Gschwandtner, Thomas Fahringer (University of Innsbruck), Gianmarco Accordi, Davide Gadioli, Emanuele Vitali, Gianluca Palermo (Politecnico di Milano), Andrea Rosario Beccari (Dompe), Daniele Cesarini, Federico Ficarelli (CINECA)

EuroHPC Joint Undertaking
MUR Ministero dell'Università e della Ricerca
DFG Deutsche Forschungsgemeinschaft