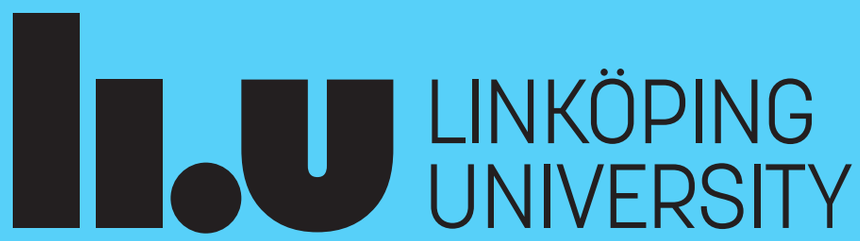
EXCESS workshop
(From HLPP 2016, updated)

August 26, 2016

SkePU 2

Flexible and Type-Safe Skeleton Programming for Heterogeneous Parallel Systems

August Ernstsson Lu Li Christoph Kessler
{firstname.lastname}@liu.se
Linköping University, Sweden



Overview

Background

- Skeleton Programming
- SkePU 1

Results

- SkePU 2
- Readability Survey
- Performance Evaluation
- Conclusions

LINKÖPING UNIVERSITY

Skeleton Programming

Skeleton Programming :: Motivation

Programming parallel systems is hard!

- Resource utilization
- Synchronization
- Communication
- Memory consistency
- Different hardware architectures
- Heterogeneity

Skeleton Programming :: Introduction

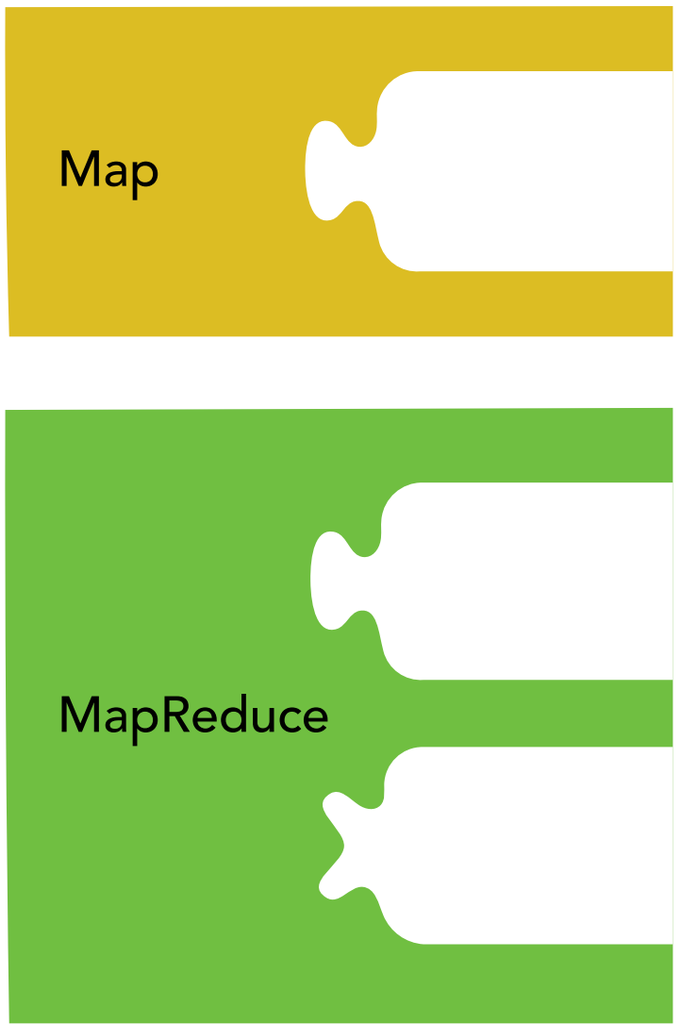
Skeleton programming (algorithmic skeletons)

- A high-level parallel programming concept
- Inspired by functional programming
- Generic computational patterns
- Abstracts architecture-specific issues

Skeletons

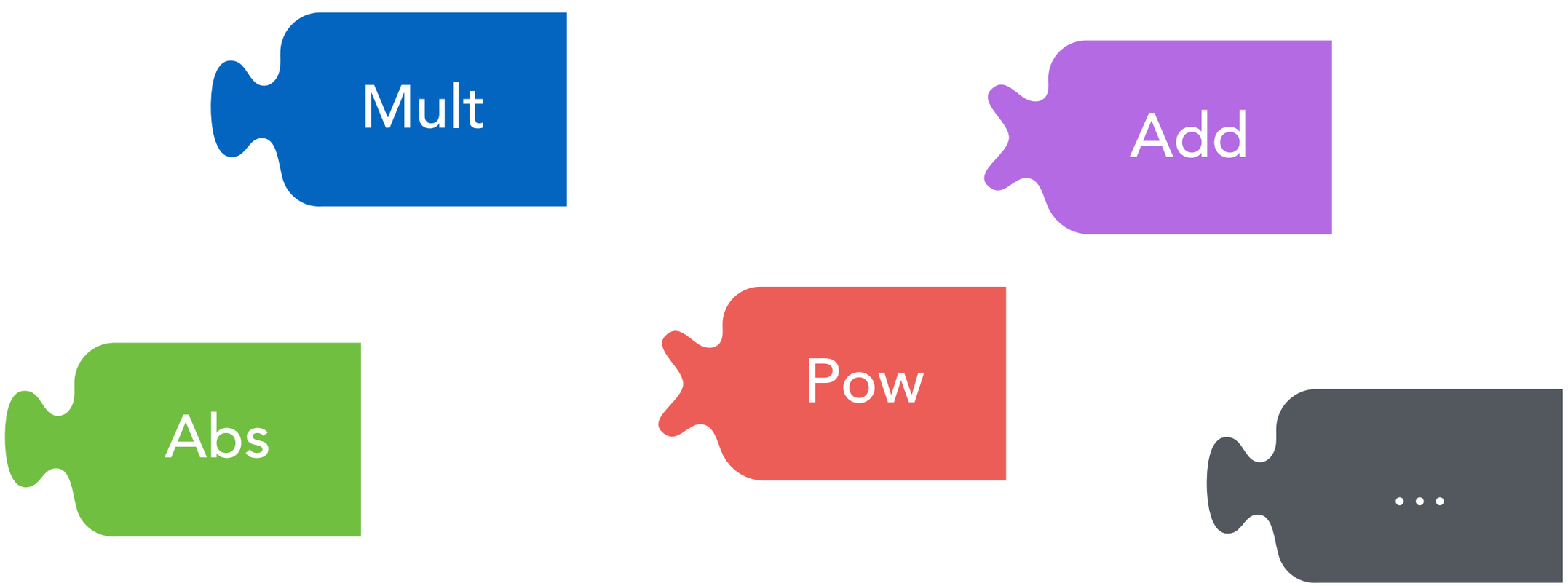
Parametrizable higher-order constructs

- Map
- Reduce
- MapReduce
- Scan
- and others



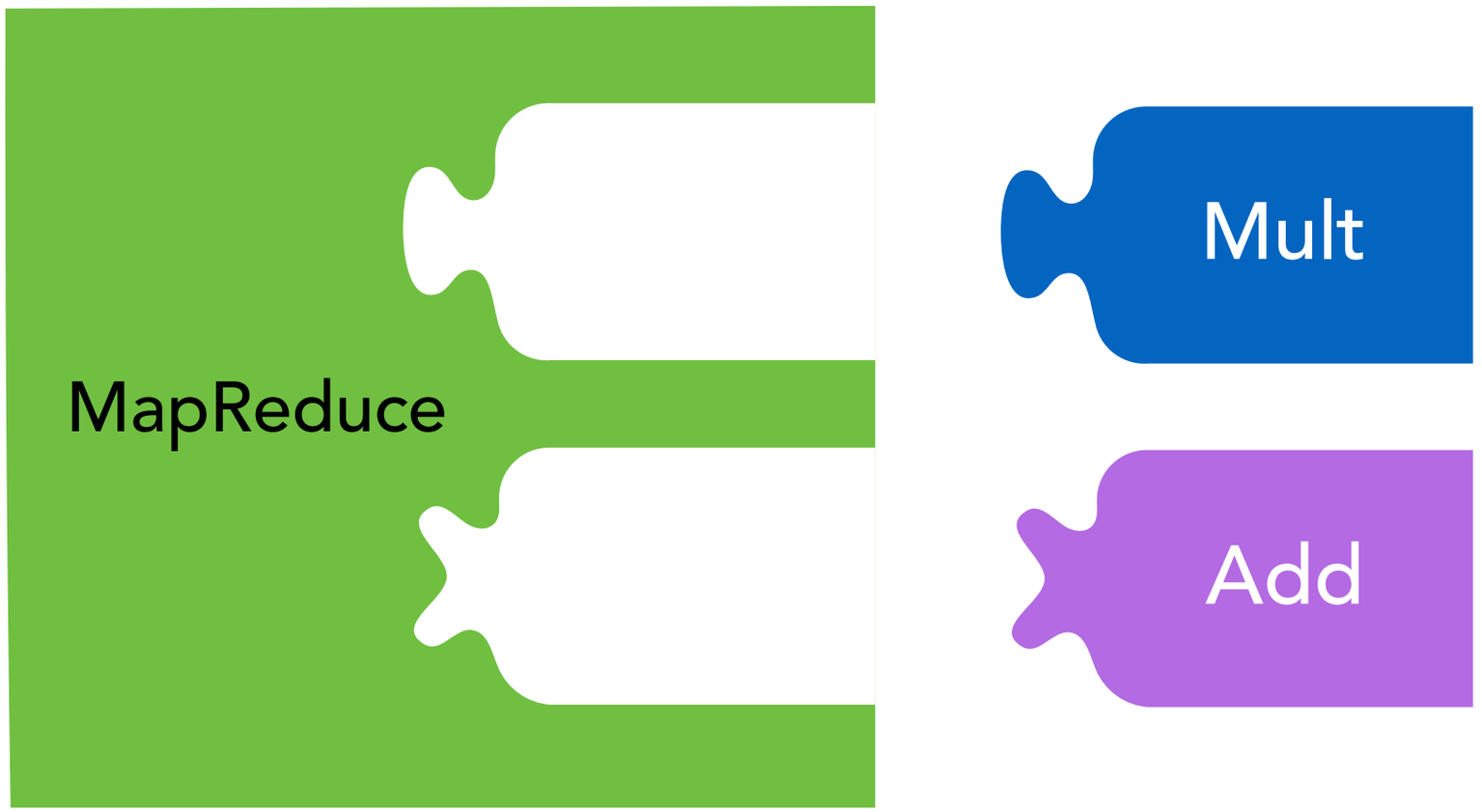
User functions

User-defined operators



Skeleton parametrization example

Dot product operation



SkePU

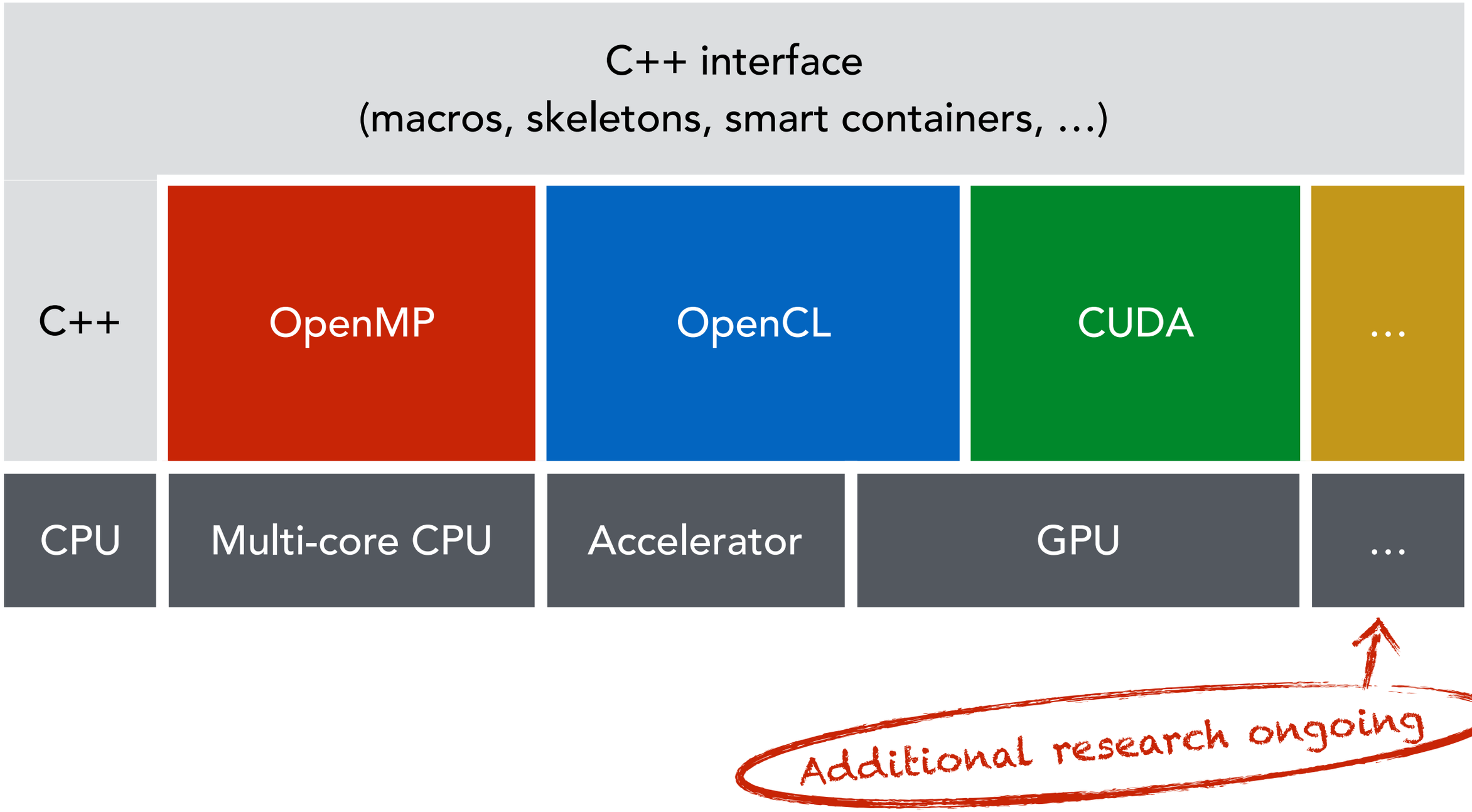
SkePU :: Features

✓ Efficient parallel algorithms

✓ Memory management and data movement

✓ Automatic backend selection and tuning

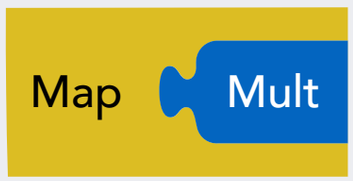
SkePU :: Architecture



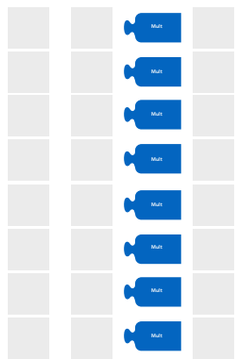
SkePU :: Syntax

```
BINARY_FUNC(add, int, a, b,
   return a + b;
)
```


```
Map<add> vec_sum(new add);
```



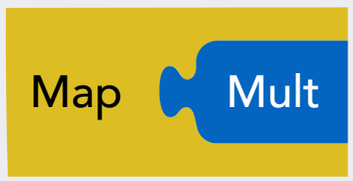

```
vec_sum(v1, v2, result);
```



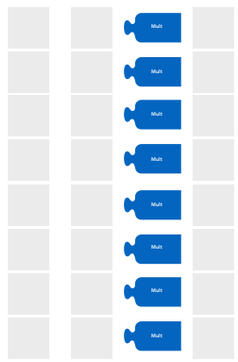
SkePU :: Syntax

```
BINARY_FUNC_CONSTANT(add, int, int, a, b, m,
  return (a + b) % m;
)
```


```
Map<add> vec_sum(new add);
```




```
vec_sum.setConstant(5);
vec_sum(v1, v2, result);
```



SkePU :: Limitations

- ✗ Non-intuitive macro syntax
- ✗ Type-unsafe user functions
- ✗ Constrained skeleton signatures

LINKÖPING UNIVERSITY

SkePU 2

SkePU 2 :: Introduction

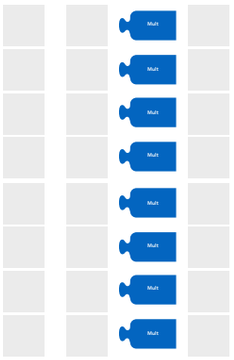
- Builds on the SkePU 1 runtime and algorithms
- New, more **native**-looking interface (API)
- Extra **source-to-source translation** step
- Based on **Clang** compiler front-end libraries

SkePU 2 :: Syntax

```
int add(int a, int b)
{
    return a + b;
}
```


```
auto vec_sum = Map<2>(add);
```


```
vec_sum(result, v1, v2);
```

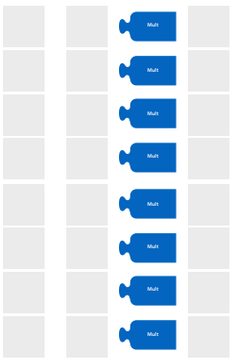


SkePU 2 :: Syntax

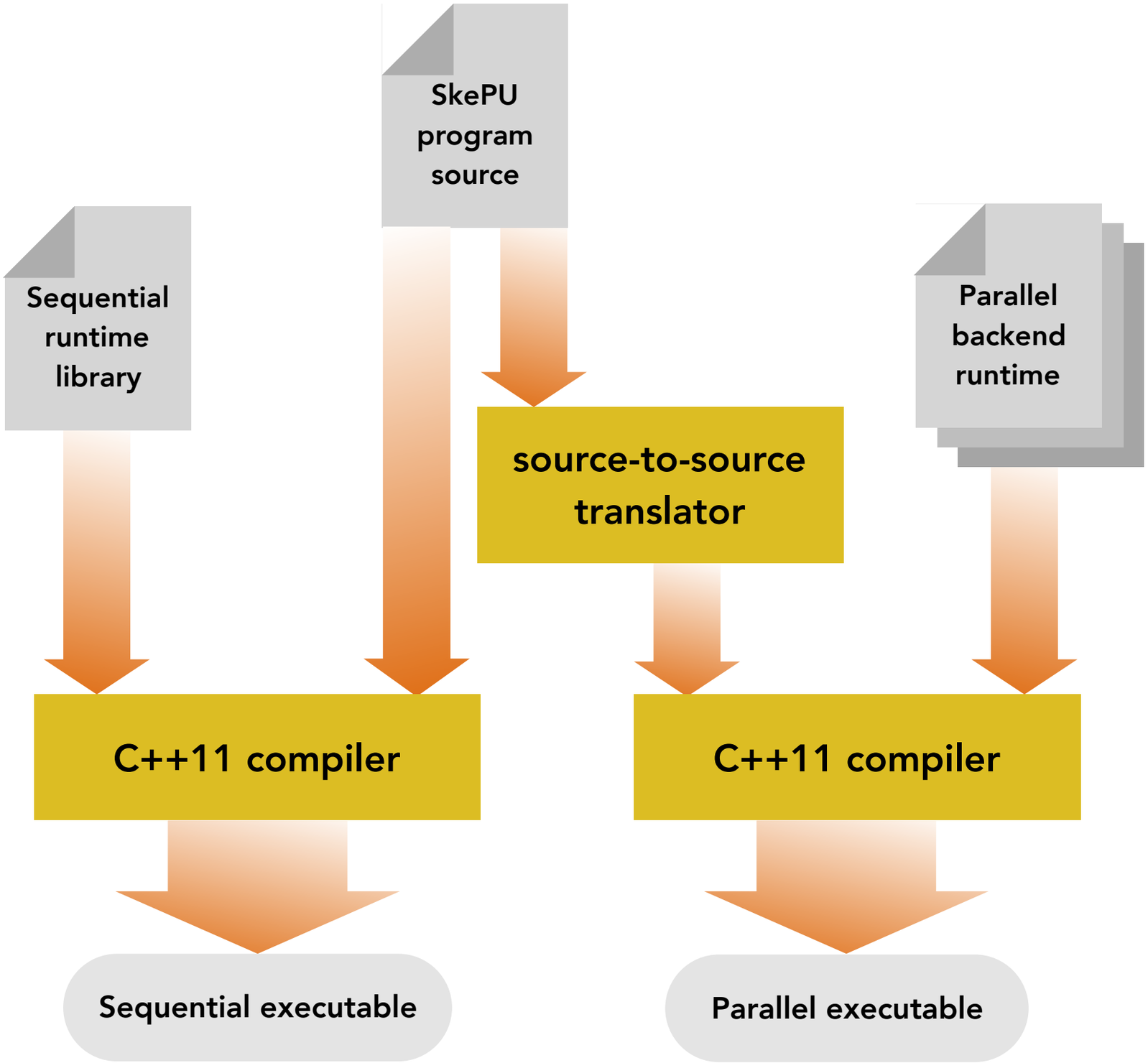
```
int add(int a, int b, int m)
{
    return (a + b) % m;
}
```


```
auto vec_sum = Map<2>(add);
```


```
vec_sum(result, v1, v2, 5);
```



SkePU 2 :: Architecture



SkePU 2 :: Flexibility

- **Variable arity** on Map and MapReduce skeletons
- **Index** argument (of current Map'd container element)
- **Uniform** arguments
- Smart container arguments accessible **freely** inside user function
  - **Read**-only / **write**-only / **read-write** copy modes
- User function **templates**

SkePU 2 :: Advanced Example

```
template<typename T>
T abs(T input)
{
   return input < 0 ? -input : input;
}
```

Templates

Readonly, no copy back

```
template<typename T>
T mvmult(Index1D row, const Matrix<T> m, const Vector<T> v)
{
   T res = 0;
   for (size_t i = 0; i < v.size; ++i)
      res += m[row.i * m.cols + i] * v[i];

   return abs(res);
}
```

Chained user functions

Type safety test-case

Reduce skeleton with **unary** user function

SkePU 1, at **run time**

```
[SKEPU_ERROR] Wrong operator type!
              Reduce operation require binary user function.
```

SkePU 2, at **compile time**

```
error: no matching function for call to 'Reduce'
                        auto globalSum = Reduce(plus_f);
                                         ^~~~~~~~~~~~~~
note: candidate template ignored: failed template argument deduction
   Reduce(T(*red)(T, T))
```

SkePU 2 :: Experimental Features

- User function **specialization** for backends
  - Extends SkePU for multi-variant components
- "Call" skeleton
- Custom **types**
- **Chained** user functions
- In-line **lambda** syntax for user functions

SkePU 2 :: Lambda Syntax

```
int add(int a, int b)
{
   return a + b;
}

auto vec_sum = Map<2>(mult);

// ...

vec_sum(result, v1, v2);
```

SkePU 2 :: Lambda Syntax

```
auto vec_sum = Map<2>([](int a, int b)
{
   return a + b;
});

// ...

vec_sum(result, v1, v2);
```

LINKÖPING UNIVERSITY

Readability Survey

Readability Survey

- Survey was made on a development version of SkePU 2 with a slightly **different syntax**
- Main difference: Used **C++11 attributes**
  - `[[skepu::userfunction]]` on user functions
  - `[[skepu::instance]]` on skeleton instances
- **Reason:** Guide the source-to-source translator and generate better error messages

Readability :: Simple Example

1

```
BINARY_FUNC(sum, int, a, b,
    return a + b;
)

Vector<float> vector_sum(Vector<float> &v1, Vector<float> &v2)
{
    Map<sum> vsum(new sum);
    Vector<float> result(v1.size());

    vsum(v1, v2, result);
    return result;
}
```

2

```
[[skepu::userfunction]]
float sum(float a, float b)
{
    return a + b;
}

Vector<float> vector_sum(Vector<float> &v1, Vector<float> &v2)
{
    auto vsum [[skepu::instance]] = Map<2>(sum);
    Vector<float> result(v1.size());

    vsum(result, v1, v2);
    return result;
}
```

Readability :: Complex Example

1

```
UNARY_FUNC_CONSTANT(kth_term, float, float, k, x,
    float temp_x = pow(x, k);
    int sign = ((int)k % 2 == 0) ? -1 : 1;
    return sign * temp_x / k;
)

BINARY_FUNC(plus, float, a, b,
    return a + b;
)

GENERATE_FUNC(init, float, float, index, seed,
    return index + 1;
)
```

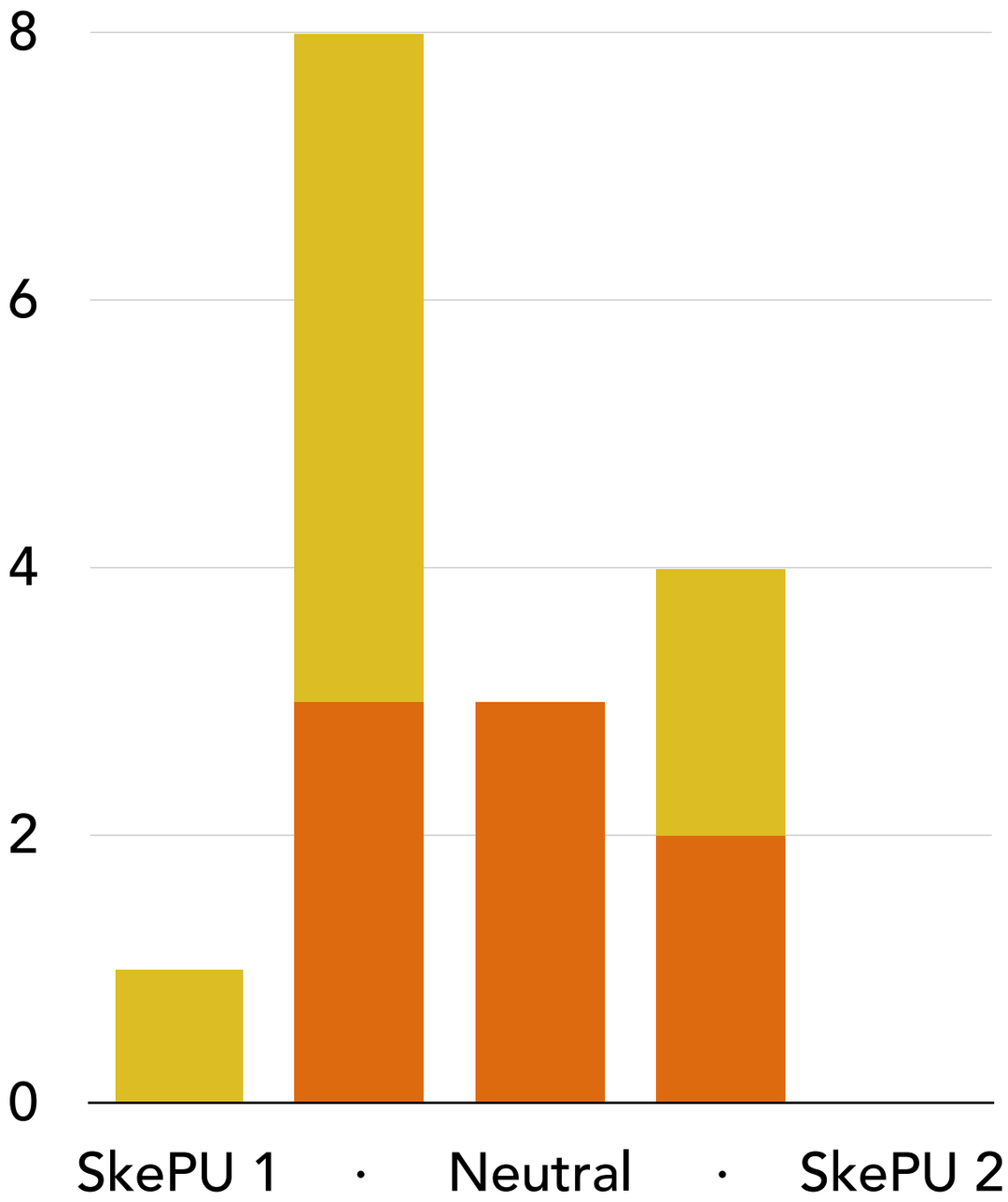
```
float taylor_approx(float x, size_t N)
{
    skepu::MapReduce<kth_term, plus>
        taylor(new kth_term, new plus);

    skepu::Generate<init>
        vec_init(new init);

    taylor.setConstant(x);

    skepu::Vector<float> terms(N);
    vec_init(N, terms);

    return taylor(terms);
}
```

2

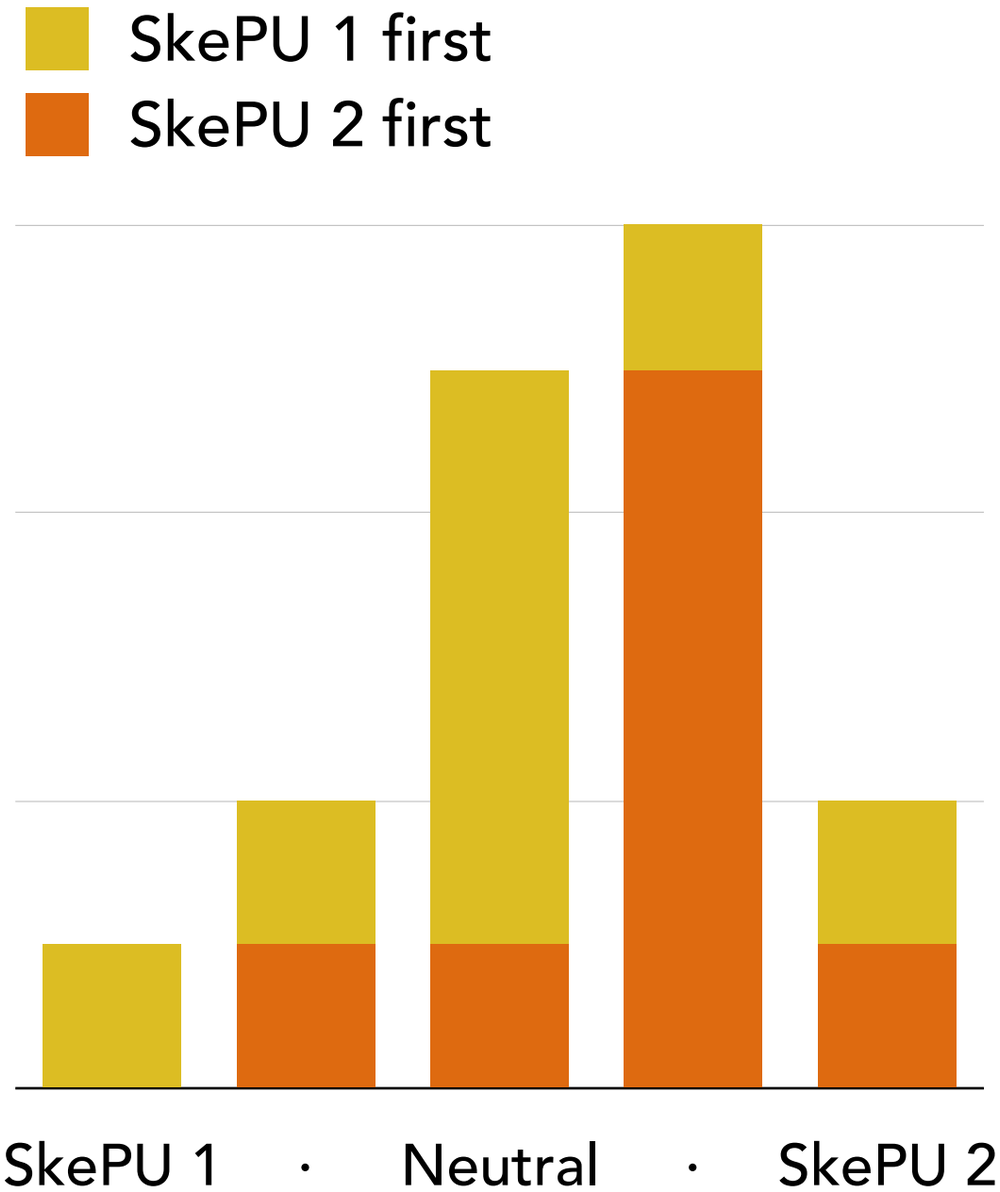
```
[[skepu::userfunction]]
float kth_term(skepu2::Index1D index, float x)
{
    int k = index.i + 1;
    float temp_x = pow(x, k);
    int sign = (k % 2 == 0) ? -1 : 1;
    return sign * temp_x / k;
}

[[skepu::userfunction]]
float plus(float a, float b)
{
    return a + b;
}

float taylor_approx(float x, size_t N)
{
    auto taylor [[skepu::instance]]
        = skepu2::MapReduce<0>(kth_term, plus);

    taylor.setDefaultSize(N);

    return taylor(x);
}
```

Readability :: Results

First example (simple)



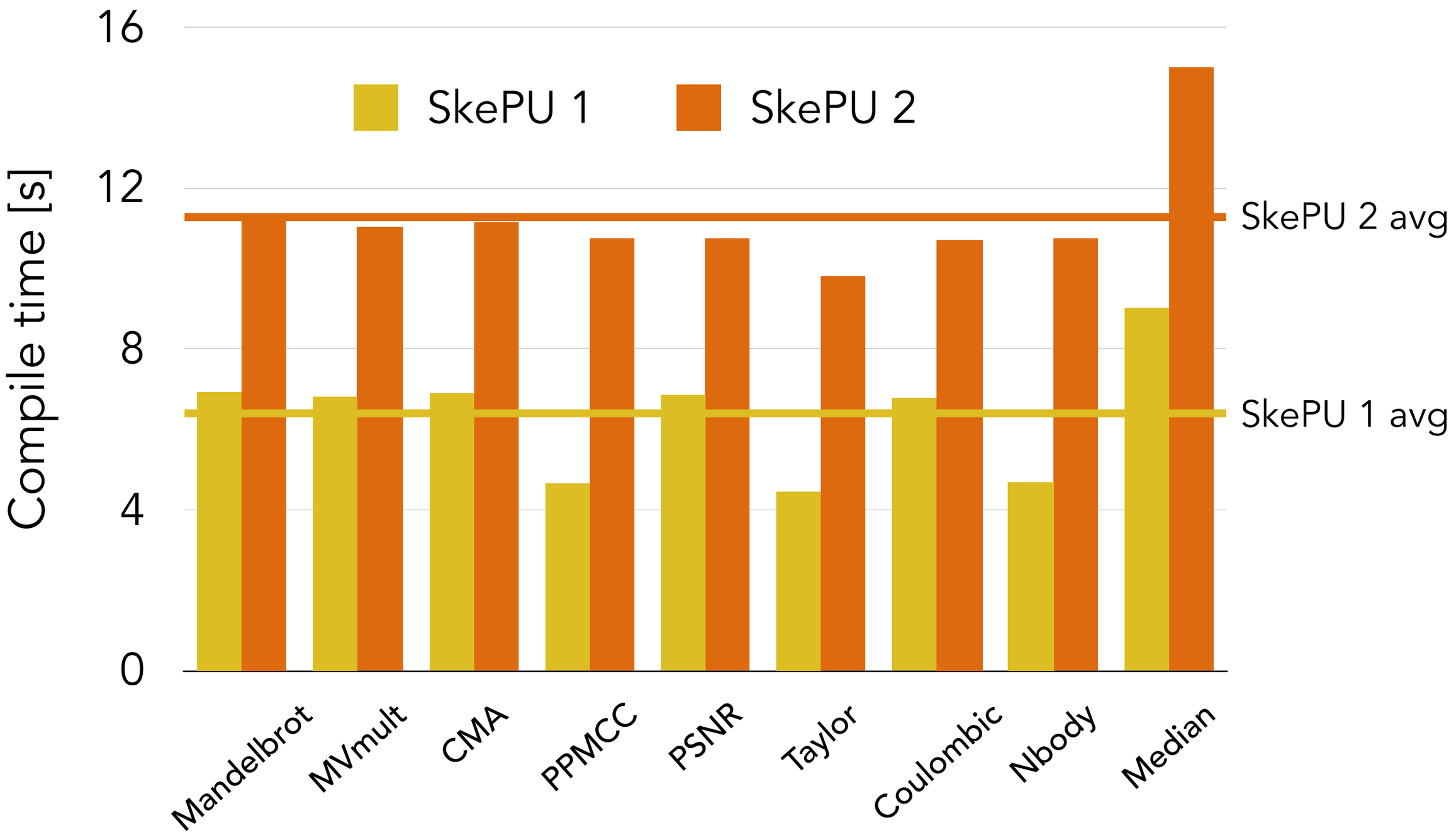
Second example (complex)



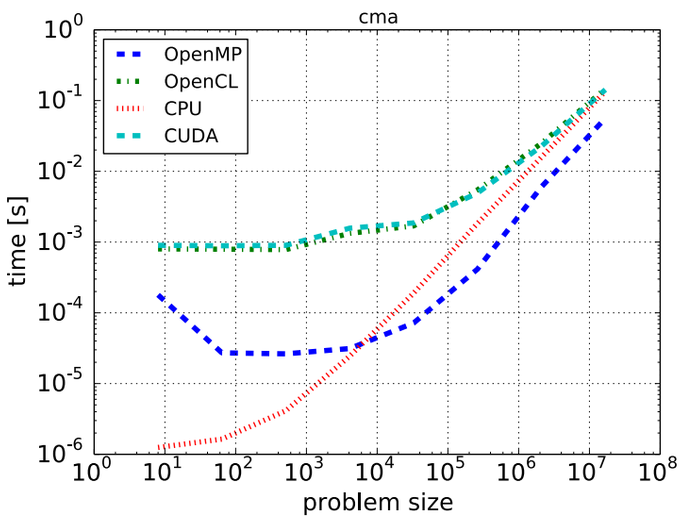
LINKÖPING UNIVERSITY

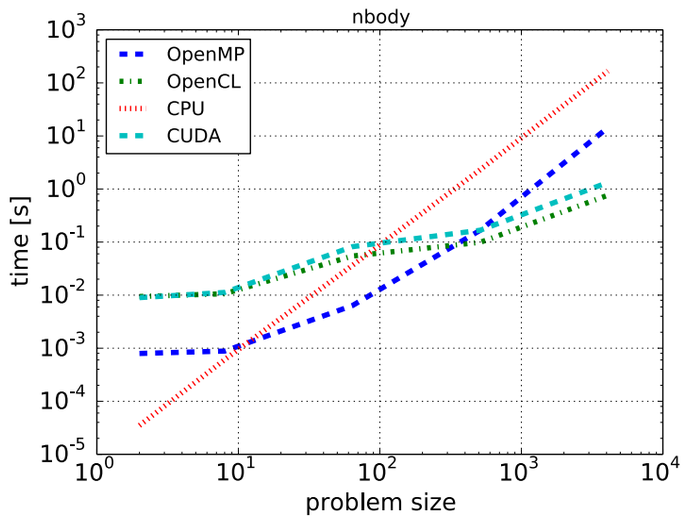
Performance Evaluation

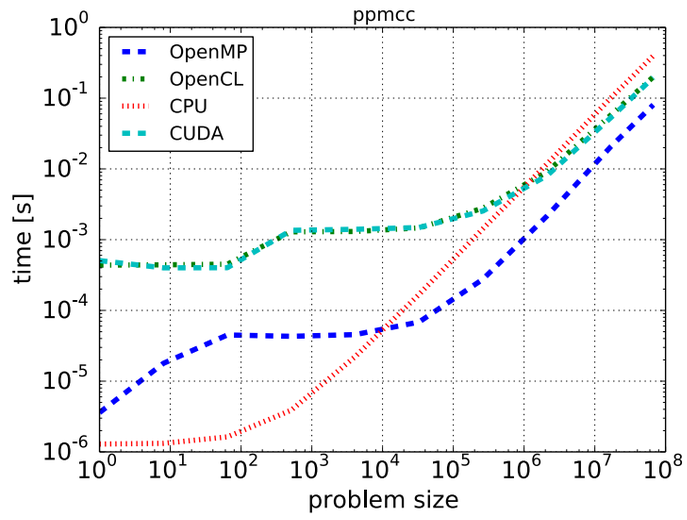
Performance :: Compile Time

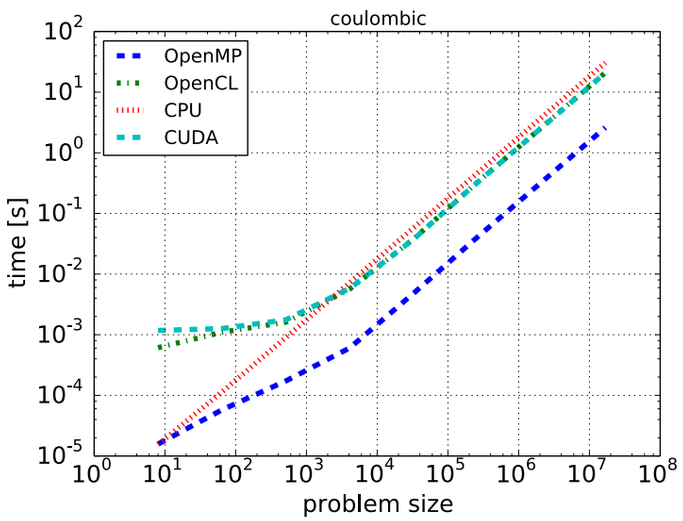


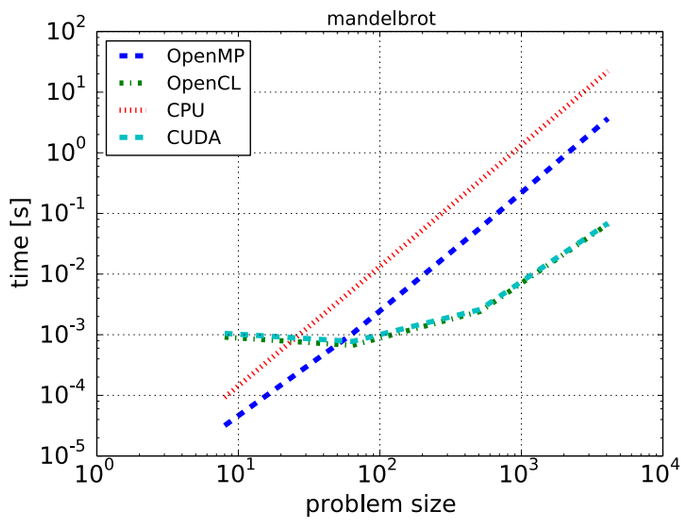
Performance :: Backends

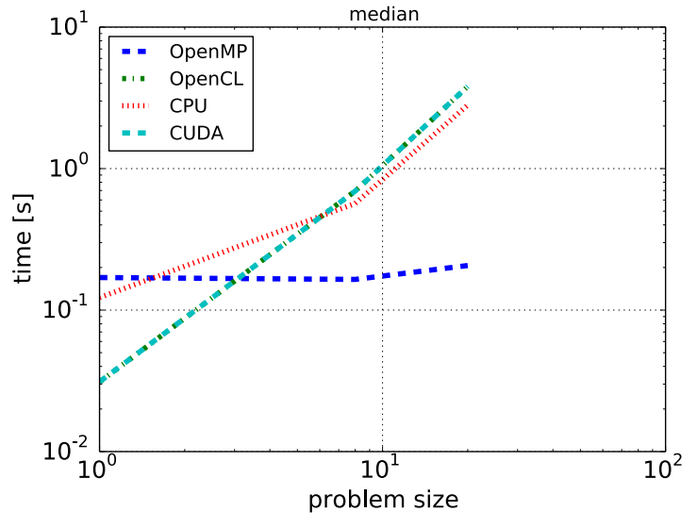






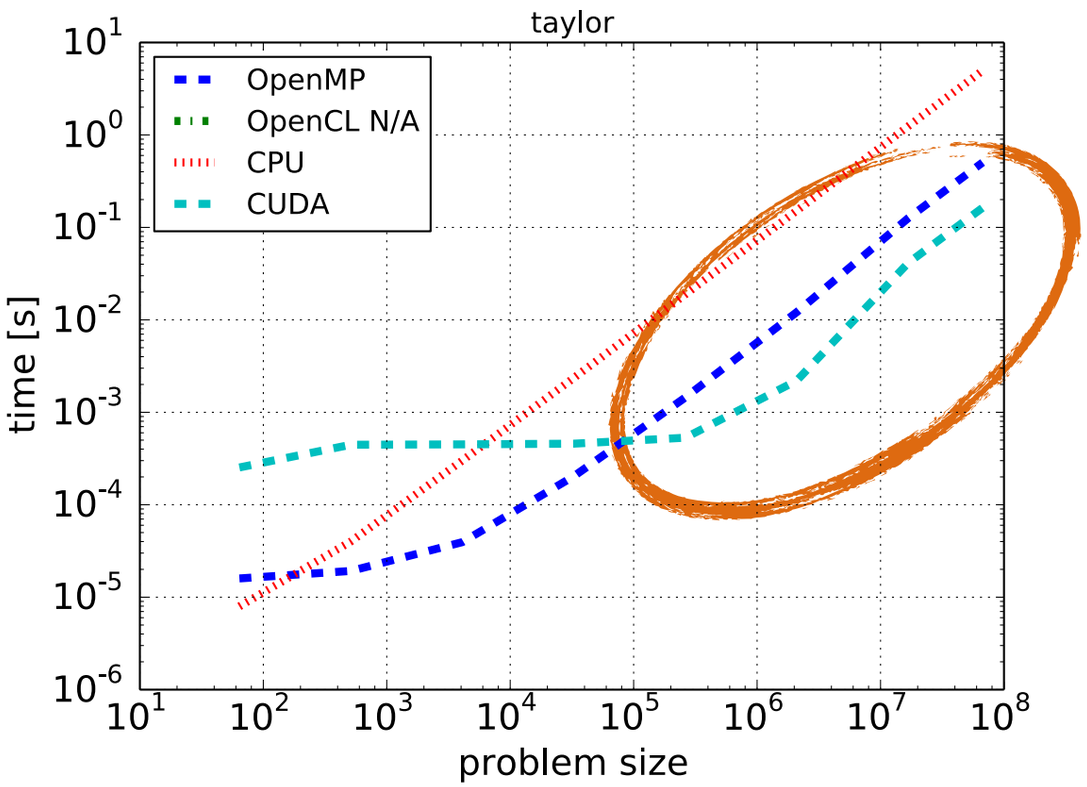




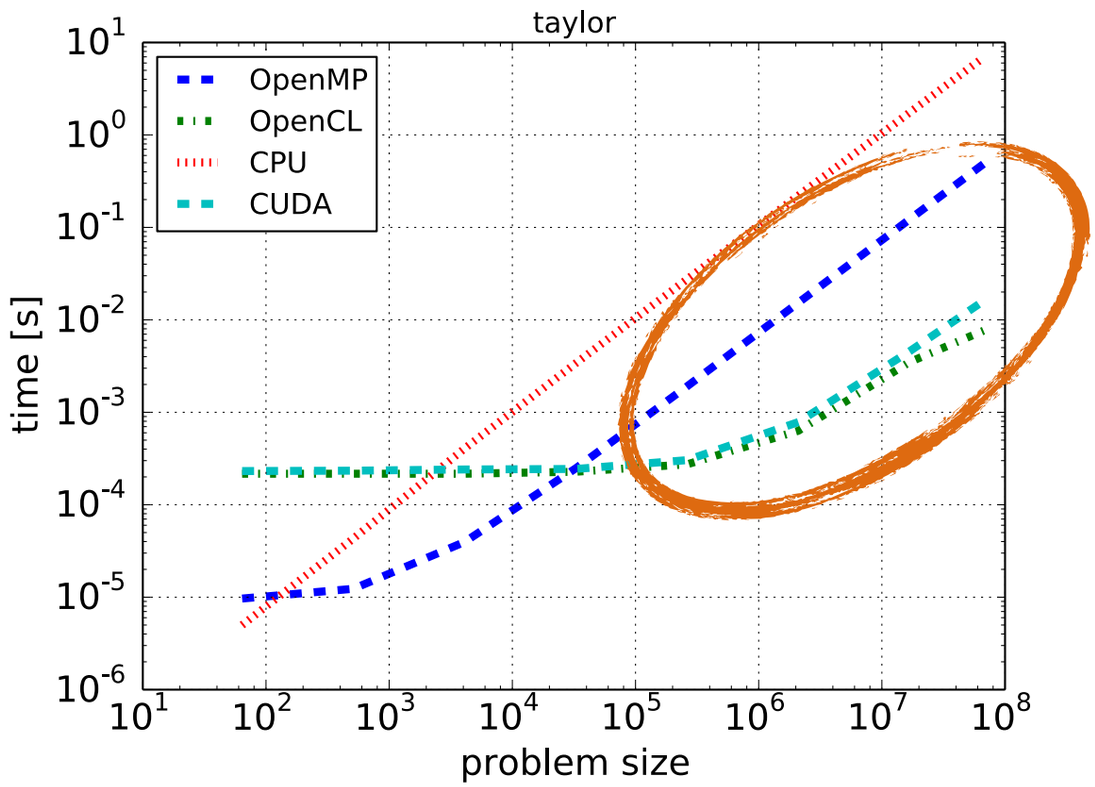


Performance :: Versions

SkePU 1.2



SkePU 2



Conclusions

Conclusions

- **SkePU 2 advancements**
  - \+ A native-looking and flexible interface
  - \+ Better type safety
  - \+ Possibility for more efficient algorithms
- **Current limitations**
  - – Needs more performance evaluation
  - – Some SkePU 1 features are not available
  - – ~~C++11 attributes may be unfamiliar to users~~

Conclusions :: Availability

SkePU2

will be distributed as **open source** soon.

Check the website at:

http://www.ida.liu.se/labs/pelab/skepu/