17th IEEE eScience 2021
INTERNATIONAL CONFERENCE

September 21, 2021

SkePU

https://skepu.github.io

Portable Programming of Heterogeneous Parallel Systems with SkePU

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

Updated slides at: https://skepu.github.io/tutorials/escience21

LINKÖPINGS UNIVERSITET

exa2pro
Enhancing Programmability and boosting Performance Portability For Exascale Computing Systems

Acknowledgments

The latest version of SkePU (https://skepu.github.io)
is joint work with **August Ernstsson** and **Johan Ahlqvist**,
and was partly funded by EU H2020 project **EXA2PRO**.

Details can be found in **August Ernstsson's Licentiate thesis** https://doi.org/10.3384/lic.diva-170194
and in the following open-access article:

August Ernstsson, Johan Ahlqvist, Stavroula Zouzoula, Christoph Kessler:
"SkePU 3: Portable High-Level Programming of Heterogeneous Systems and HPC Clusters." *International Journal of Parallel Programming*, Springer, May 2021 (online), print version to appear. https://link.springer.com/article/10.1007/s10766-021-00704-3

Many **others** have contributed to SkePU over the years, too,
see the SkePU web page for acknowledgments.

Earlier funding sources include EU FP7 PEPPHER and EXCESS, as well as CUGS and SeRC.
Access to HPC resources used in the reported work was provided by SNIC / NSC Linköping.

LINKÖPINGS UNIVERSITET

exa2pro

Agenda

Introduction to Skeleton Programming and SkePU Overview

SkePU Fundamentals

SkePU API

Examples

SkePU Advanced Topics

September 21, 2021

SkePU

https://skepu.github.io

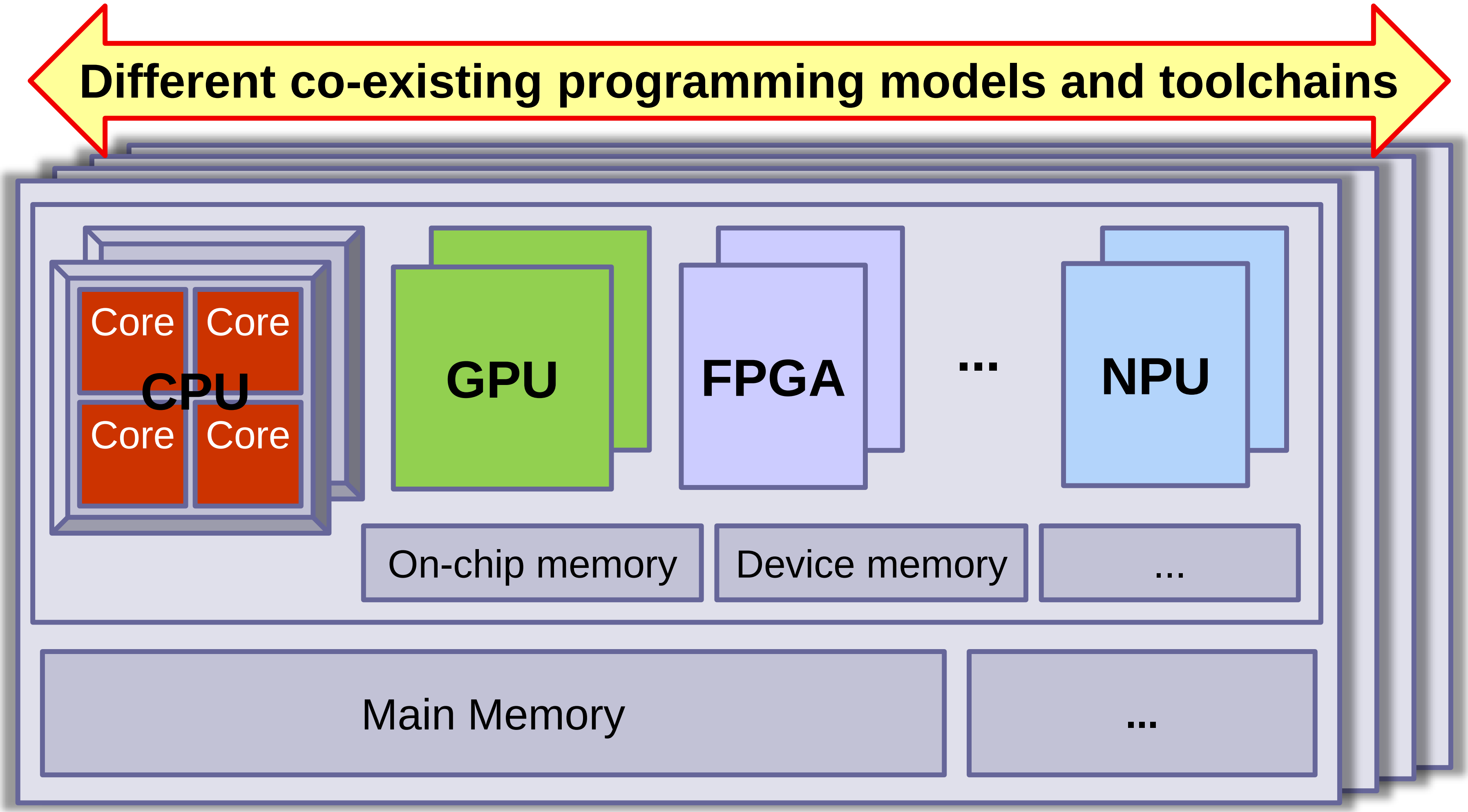
Introduction to Skeleton Programming and SkePU Overview

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

LINKÖPINGS UNIVERSITET

exa2pro
Enhancing Programmability and boosting Performance Portability For Exascale Computing Systems

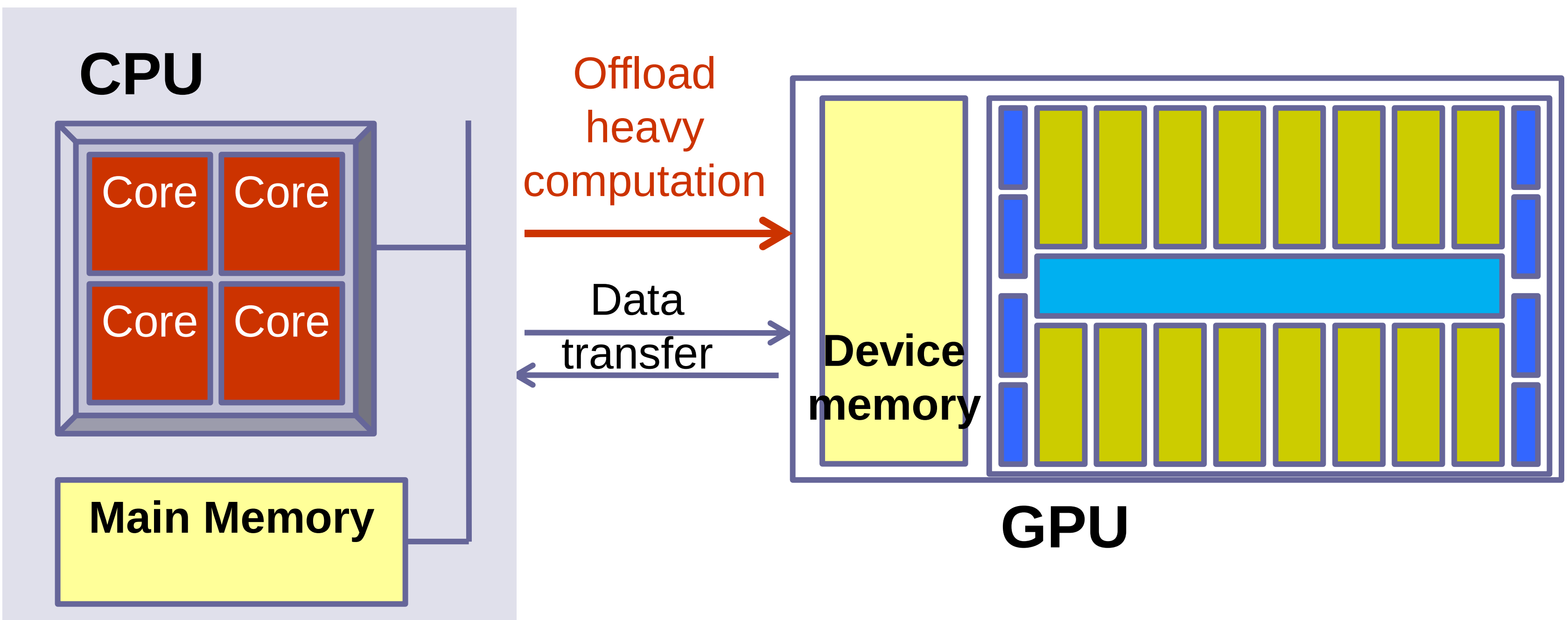
Parallelism and Heterogeneity are here to stay ...



→ ***Software*** **Productivity ??**

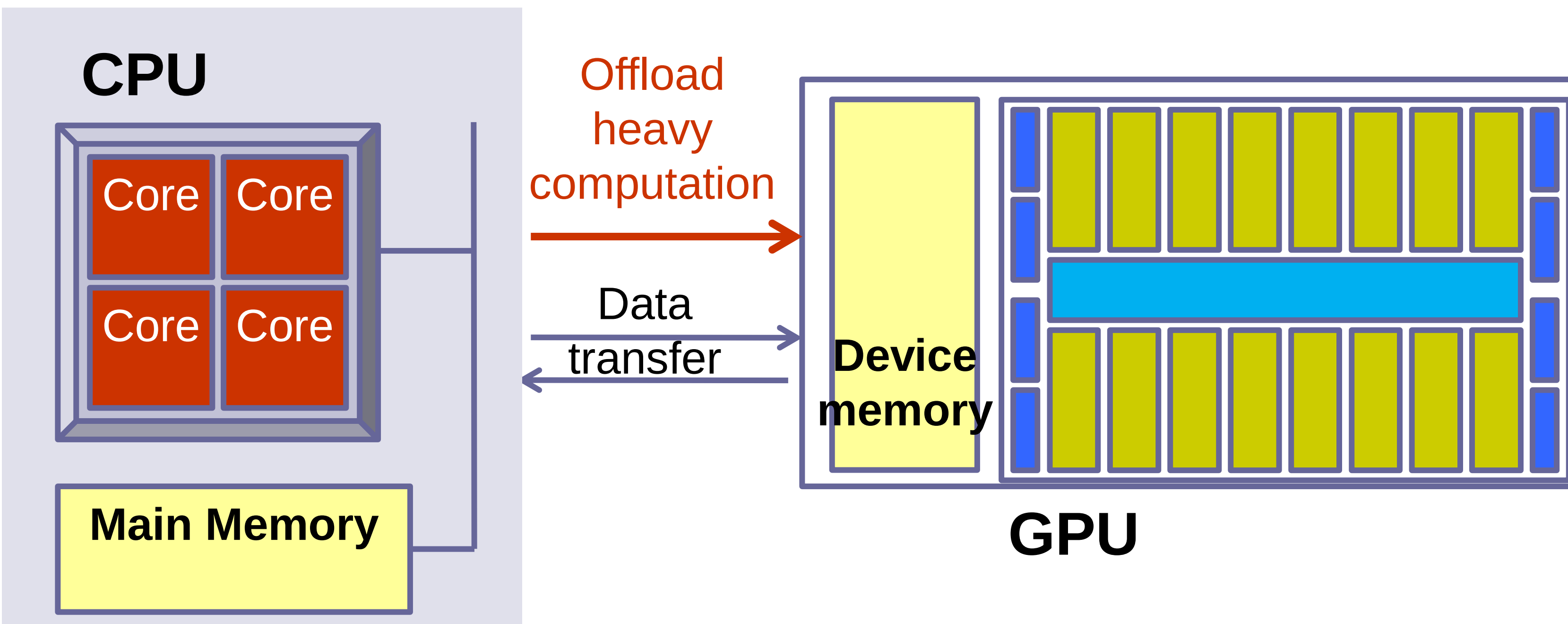
Example: GPU-Accelerated Systems

Image source: Microway



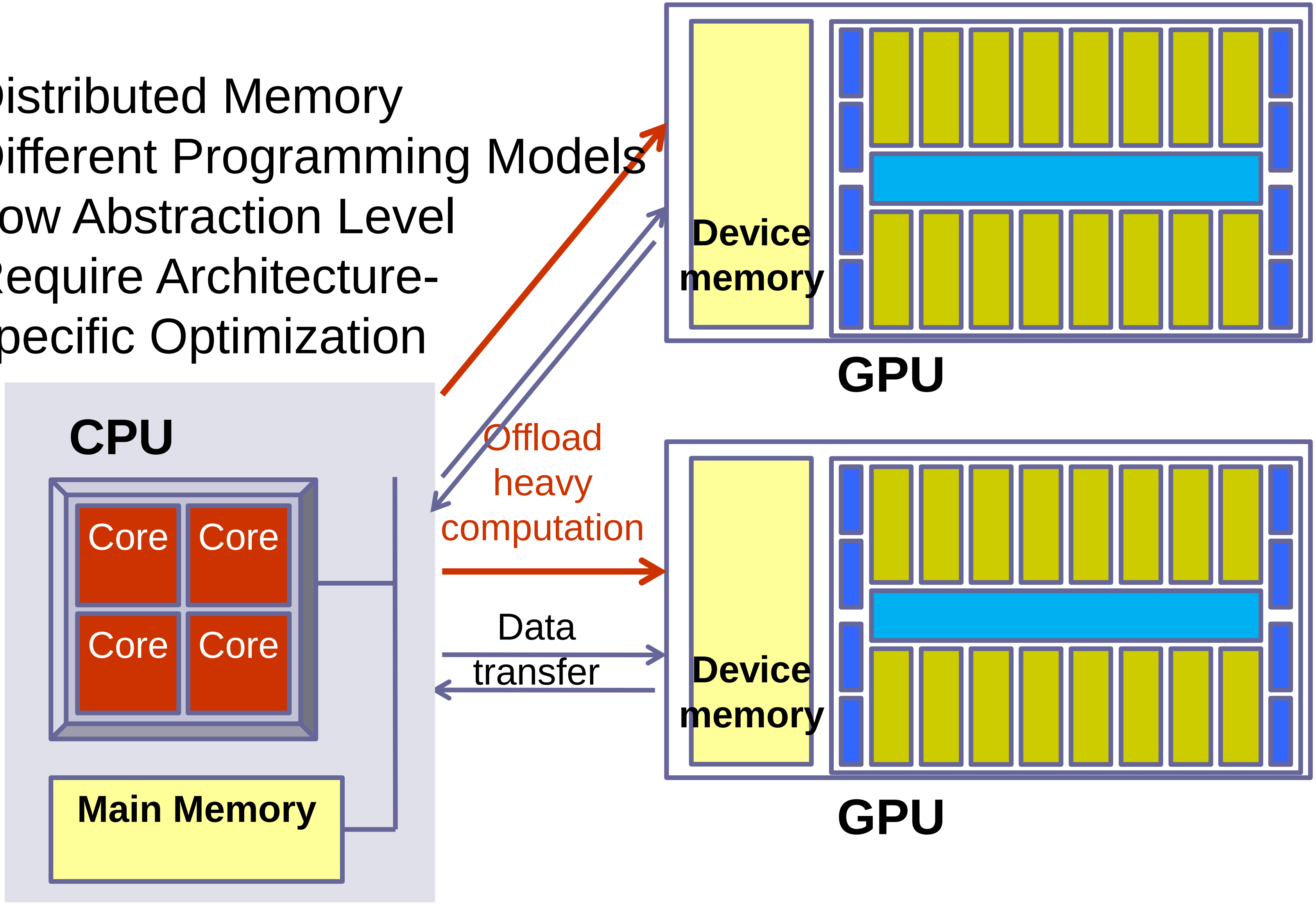
Example: GPU-Accelerated Systems

- ☹ Distributed Memory
- ☹ Different Programming Models
- ☹ Low Abstraction Level
- ☹ Require Architecture-specific Optimization



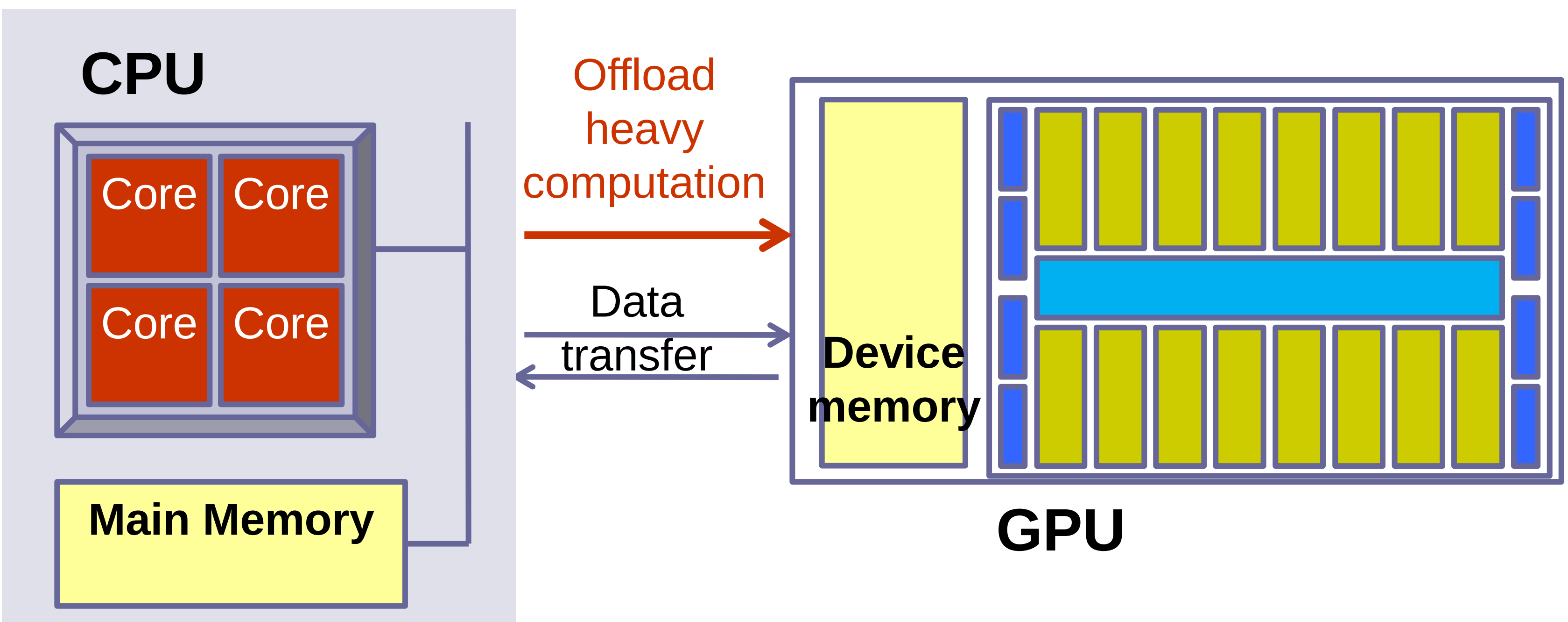
Example: GPU-Accelerated Systems

- ☹ Distributed Memory
- ☹ Different Programming Models
- ☹ Low Abstraction Level
- ☹ Require Architecture-specific Optimization



Programming of GPU-accelerated Systems … with OpenCL™

- Code portability ☺
- Programmability ☹ (low level)
- Performance portability ☹ (requires reoptimization)

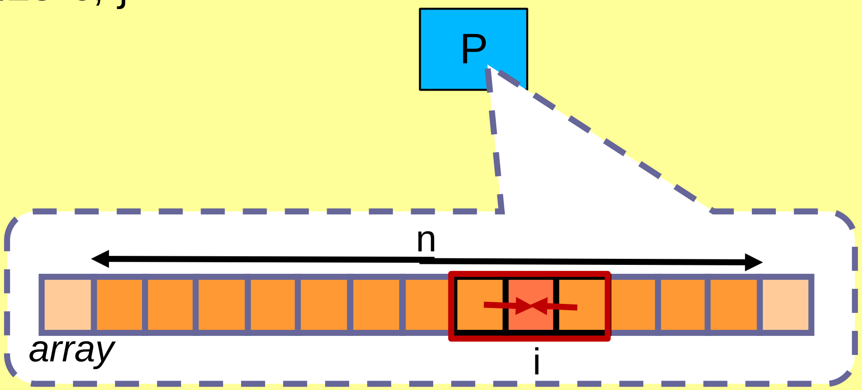


Programming of Clusters … with MPI

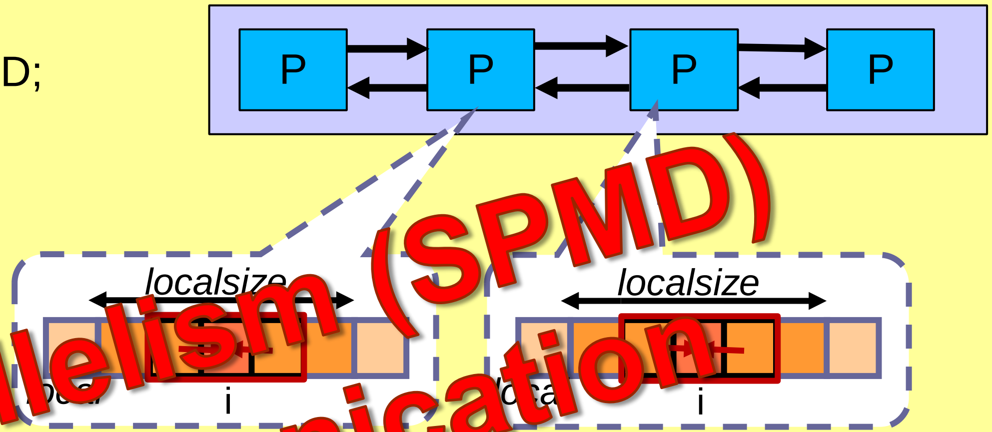
- **Code portability** ☺
- **Programmability** ☹ (low level)
- **Performance portability** ☹

Example:
1D filter in C

```
...
float filter (float a, b, c) { return 0.25*a + 0.5*b + 0.25*c; }
…
void main ( int argc, char *argv[] )
{
  float *array ( n+2 );
  float *tmp ( n+2 );
   ...
  while ( globalerr > 0.1 )  {
      for (i=1; i<=n; i++)
          tmp[i] = filter( array[i-1], array[i], array[i+1] );
      globalerr = 0.0;
      for (i=1; i<=n; i++)
           globalerr = fmax (globalerr, fabs( array[i] – tmp[i] ));
      for (i=1; i<=n; i++)
          array[i] = tmp[i];
 }
}
```



Example:
1D filter in C + MPI




```
...
void main ( int argc, char *argv[] ) {
   MPI_Comm com = MPI_COMM_WORLD;
   MPI_Init ( &argc, &argv );
   MPI_Comm_size ( com, &np );
   MPI_Comm_rank ( com, &me );
   ...
   localsize = (int) ceil ( (float) n / np );
   FloatArray local( localsize + 2);
   ...
   while ( globalerr > 0.1 ) {
      if (me>0)       MPI_Send ( local+1, 1, MPI_FLOAT, left_neighbor, 10, com );
      if (me<np-1)  MPI_Send ( last, 1, MPI_FLOAT, right_neighbor, 20, com );
      for (i=1; i<=localsize; i++)
         tmp[i] = filter( local[i-1], local[i], local[i+1] );
      if (me<np-1) MPI_Recv ( tmp, 1, MPI_FLOAT, right_neighbor, 10, com, ... );
      if (me>0)       MPI_Recv ( tmp+localsize+1, 1, MPI_FLOAT, left_neighbor, 20, com, ... );
      tmp[1] = filter( local[0], local[1], local[2] );
      tmp[localsize] = filter( local[localsize-1], local[localsize], local[localsize+1] );
      localerr = 0.0;
      for (i=1; i<=localsize; i++)  localerr = fmax( localerr, fabs ( local[i]-tmp[i] ));
      MPI_Allreduce ( &localerr, &globalerr, 1, MPI_FLOAT, MPI_MAX, com );
      for (i=1; i<localsize; i++)
         local[i] = tmp[i];
   }  ...
}
```

Explicit parallelism (SPMD)

Explicit communication

Complex code

Error-prone

Platform-specific optimizations hardcoded

Our goal:

Raise the level of abstraction

*** towards portable, future-proof programs for heterogeneous parallel systems**

*** Try to make it as easy as sequential programming**

Techniques:

- Skeleton Programming
- Portable Data Abstractions (Data-Containers)

Observation

- The same (platform-specific) technique for identifying/managing parallelism, communication, synchronization... is re-applicable for all occurrences of the same **specific structure** (**pattern**) **of computation**
  - Elementwise operations on arrays
  - Stencil computations
  - Reductions
  - Scan (Prefix-op)
  - Divide-and-Conquer
  - Farming independent tasks
  - Pipelining
  - …
- Most of these have both sequential and parallel implementations
- Idea: **Reusable** (customizable) generic constructs (***skeletons***)

Data Parallelism

Given:

- Operand data-containers ***a***, ***b***, … *with* n elements each, e.g., array(s) $\boldsymbol{a} = (a_1,...a_n)$, $\boldsymbol{b} = (b_1,...,b_n)$, …
- A side-effect-free operation f on elements of ***a***, ***b***, …
  - Any arity: 0, 1, 2 or more input operands,
    1 or more output operands

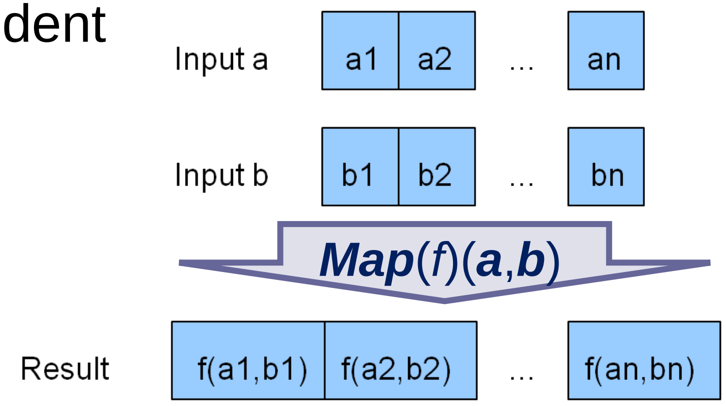
Compute: $(f(a_1,b_1,...), ..., f(a_n,b_n,...))$

Parallelizability:

- All elementwise computations are independent
- Easily partitioned into independent tasks of arbitrary granularity

Notation with higher-order function:

- $\boldsymbol{y} = \boldsymbol{Map}\ (f)\ (\boldsymbol{a}, \boldsymbol{b}, ...)$



Data Parallelism with Stencils

A generalization of *Map*:

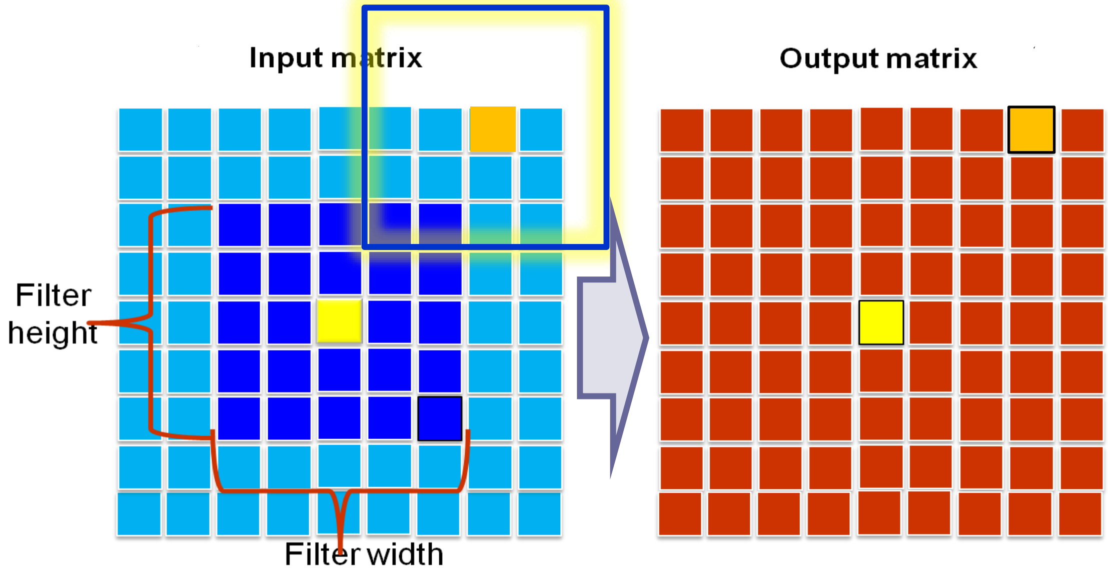
For calculating ***y***[*i*], can access elements in ***a*** within a small (constant-sized) halo **region** around element index *i*

- Access window on ***a***, e.g., for 1D region: ***a***[*i*-*K*]…***a***[*i*+*K*] for given *K*>0

For stencil computations (filters, convolutions, PDE solving, …)

- Typically, 2D, 3D or 4D operands and regions

y = ***MapOverlap*** (*filterfunction*) (***a***, …)



Further customization:

Special treatment of elements at domain boundaries (edges)
- constant padding
- duplication padding
- cyclic / toroidal padding
- ...

Update schema
- Jacobi style
- In-place, Red-Black
- ...

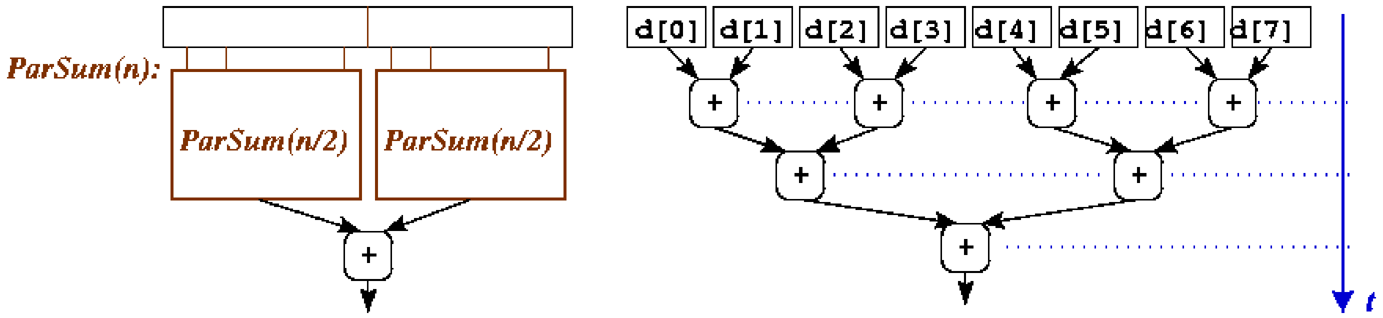
Data-Parallel Reduction

Given:

- A data container ***d*** with *n* elements,
  e.g. array ***d*** $= (d_1,...,d_n)$
- A <u>binary, associative</u> operation *op* on individual elements of ***d***
  (e.g. *add, max, bitwise-or*, ...)

Compute: $s = OP_{i=1...n}\ \mathbf{d} = d_1\ op\ d_2\ op\ ...\ op\ d_n$

Parallelizability: Exploit *associativity* of *op*



Notation with higher-order function:

- *s* = ***Reduce*** (*op*) (***a***)

Further customization:
For higher-dimensional data-containers,
e.g. matrix: total / row-wise / column-wise

MapReduce (pattern)

Common combination:

$$\mathbf{a}, \ldots \longrightarrow \boxed{\textbf{\textit{Map}}(\, f, \mathbf{a}, \ldots)} \xrightarrow{\;\boldsymbol{y}\;} \boxed{\textbf{\textit{Reduce}}(\, g, \boldsymbol{y}, \ldots)} \longrightarrow s$$

s = ***MapReduce*** (f, g) (***a***, …)

Example:

Dot product of two vectors ***a***, ***b***: $\quad s = \sum_i a_i * b_i$

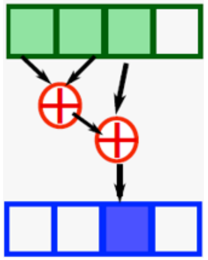
f = scalar multiplication,

g = scalar addition

Further Patterns ...

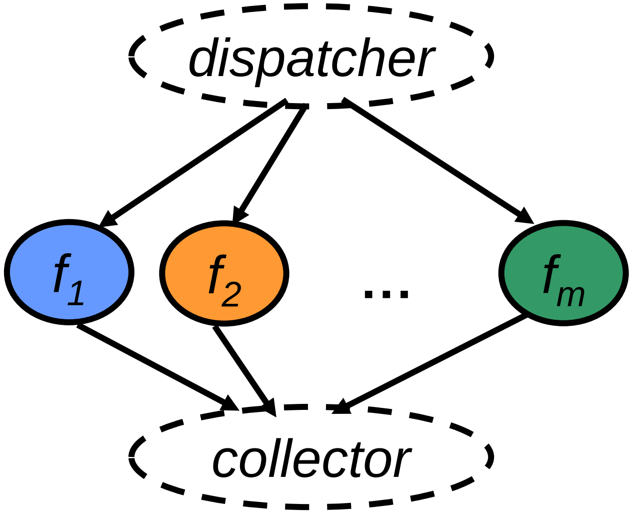
More Data-parallel patterns

- Scan (prefix-sums)
- ...



Task-parallel patterns

- Task farming
- Parallel divide-and-conquer
- ...



Stream-parallel patterns

- Pipelining
- Stream farming
- ...

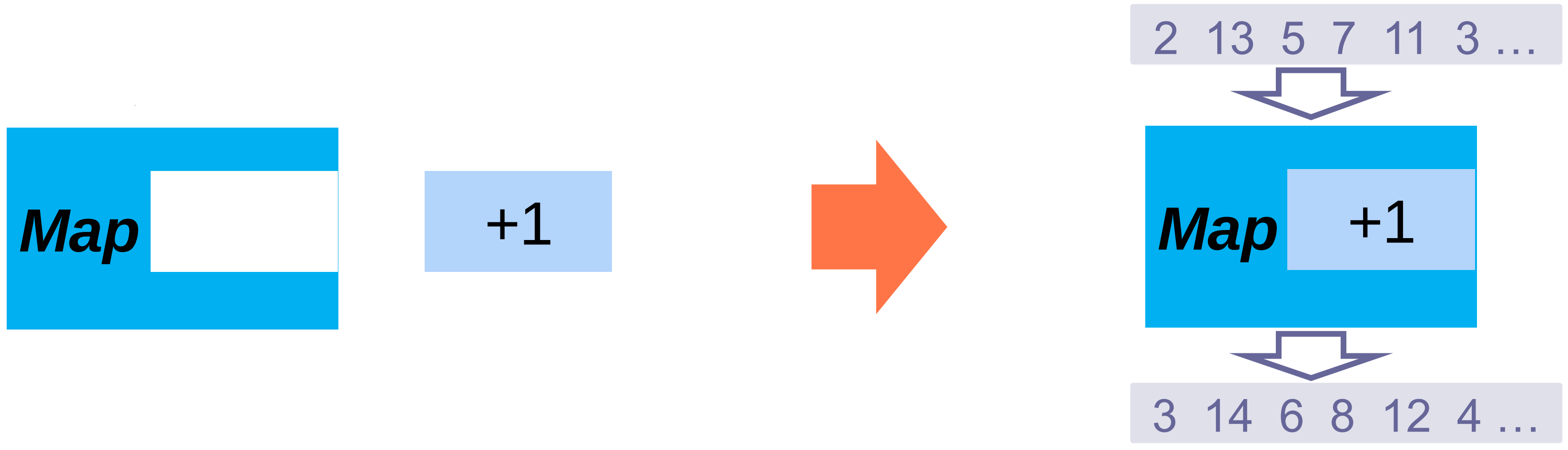
Domain-specific patterns

- ...

Skeleton / Pattern Programming [Cole'89]

- A **skeleton** is a pre-defined generic program building block based on a *higher-order function* that
  - models a common *computation / dependence pattern*

    map, reduce, scan, stencil, farm, pipe, …
  - can be *parameterized* with sequential user code
  - provides a *sequential interface*
  - can underneath have optimized implementations



Example revisited: 1D filter in C^{++} + Skeletons

```
…
float filter ( Region1D<float> r, Vec<float> w ) { return w[0]*r[-1] + w[1] * r[0] + w[2] * r[1]; }

float fabsdiff ( float a, b ) { return fabs ( a – b ); }

float idem ( float a ) { return a; }

void main ( int argc, char *argv[] )
{   …
    Vector<float> array( n );
    Vector<float> tmp ( n );
    Vector<float> weights { 0.25, 0.5, 0.25 };
    ...
    // skeleton instantiations:
    auto stencil = MapOverlap( filter );  stencil.setOverlap(1);
    auto maxchange = MapReduce( fabsdiff, fmax );
    auto copy = Map( idem );
    ...
    while ( globalerr > 0.1 )  {
        stencil( tmp, array, weights, n );
        globalerr = maxchange ( array, tmp );
        copy( array, tmp );
    }
    ...
}
```

Generic data-containers
- used as operands
- encapsulate metadata (e.g., size)

High-Level Parallel Programming with Skeletons

Skeletons + Data-containers *implement* (parallel) **algorithmic patterns**

☺ Abstraction, hiding complexity (parallelism and low-level programming)

☺ Sequential(-looking) programming interface – Parallelization for free

☺ Easier to analyze and transform

☺ Portability

😐 Enforces code (re)structuring

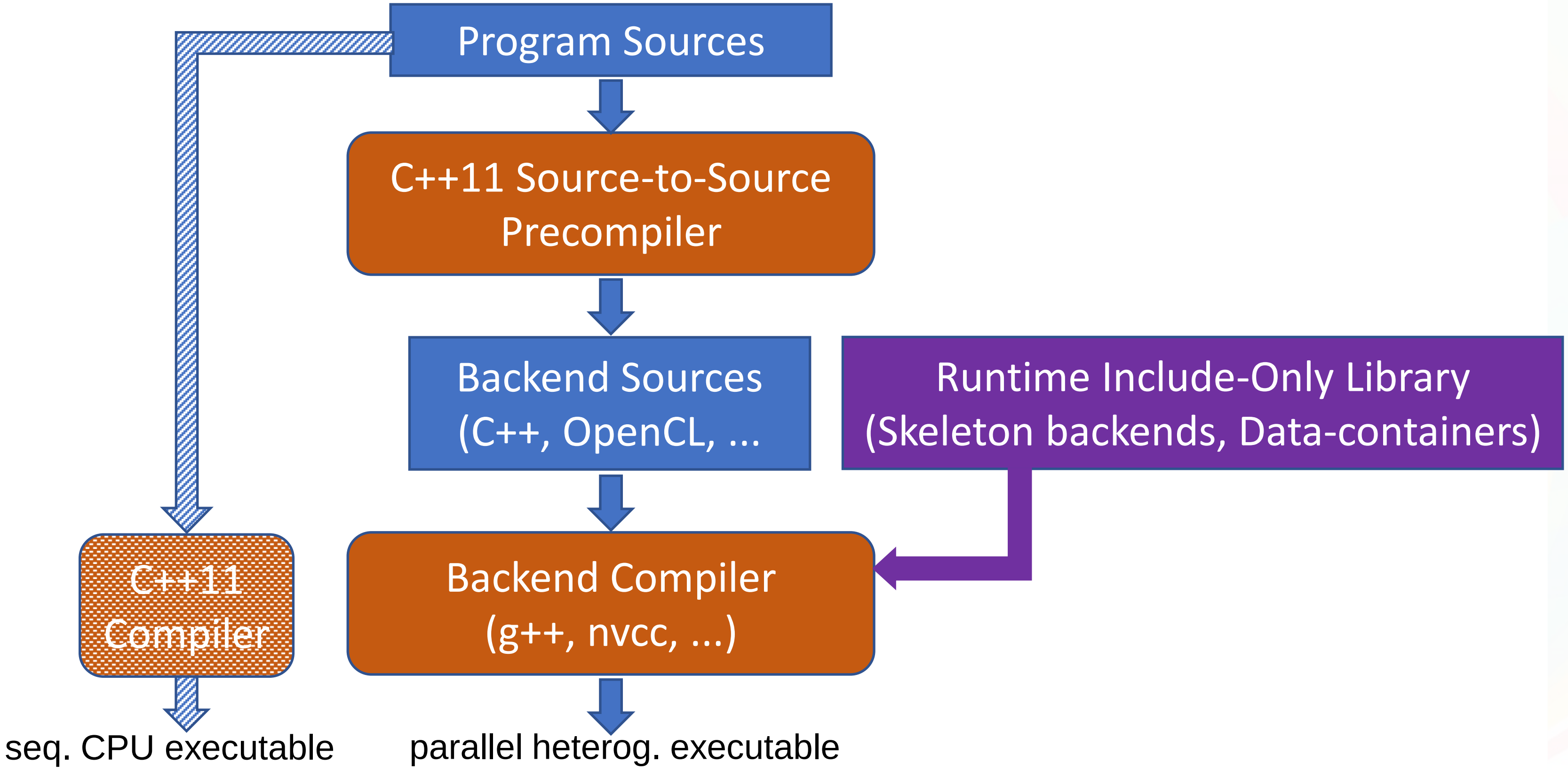
☹ Available skeleton set does not always fit

☹ May lose some efficiency compared to manual parallelization

- Idea developed since the late 1980s [Cole'89]
- Many (esp., academic) frameworks exist, mostly as libraries atop C++
  - actively researched: FastFlow, SkePU, GrPPI, LiFT, Musket, ...
- Industry has adopted skeletons, too
  - map, reduce, scan in many modern parallel programming APIs
    - e.g., Intel *TBB*: parallel for, parallel reduce, pipe
    - *Thrust, MapReduce, Spark* etc.

SkePU

- C++11 based
  - Every SkePU program is a valid C++11 program
  - Variadic template metaprogramming include library and a source-to-source pre-compiler



A. Ernstsson, L. Li, C. Kessler: **SkePU 2: Flexible and type-safe skeleton programming for heterogeneous parallel systems.** *International Journal of Parallel Programming* 46(1):62, Jan. 2018

SkePU 3

- C++11 based
  - Every SkePU program is a valid C++11 program
  - Variadic template metaprogramming include library and a source-to-source pre-compiler
- Data-parallel skeletons
  - **Map**, **Reduce**, **MapReduce**, **MapOverlap**, **MapPairs**, **MapPairsReduce**, **Scan**, ...
  - Fully variadic in elementwise, random-access and scalar operands
- STL-like data containers wrapping operand arrays
  - **Vector**<..>, **Matrix**<..>, **Tensor3**<..>, **Tensor4**<..>
  - Container windows for different operand access patterns in user functions: e.g., elementwise; **Region** (stencils); **Vec**, **Mat** (random access); **MatRow**; ...
- Multiple back-ends:
  - C, OpenMP, OpenCL, CUDA
  - New hybrid stand-alone backend [Öhberg, Ernstsson, K. 2018]
  - Multi-GPU support
  - Cluster backend for StarPU-MPI
- Tunable for time [Dastgeer *et al.*'11,'13], energy [Li, K. *J Supercomp.*'16]
- Open source: https://skepu.github.io

A. Ernstsson, J. Ahlqvist, S. Zouzoula, C. Kessler: **SkePU 3: Portable high-level programming of heterogeneous systems and HPC clusters.** *Int. J. of Parallel Programming, May* 2021

SkePU Example: Dot Product

```
#include <skepu>

float add ( float a, float b )
{
   return a + b;
}

float mult ( float a, float b )
{
   return a * b;
}

int main()
{
   … arrays A, B …
   Vector<float> u( A, N );
   Vector<float> v( B, N );

   auto dotprod = MapReduce<2>(mult, add);

   ... = dotprod( u, v );
}
```

User function (platform-independent, side-effect free)

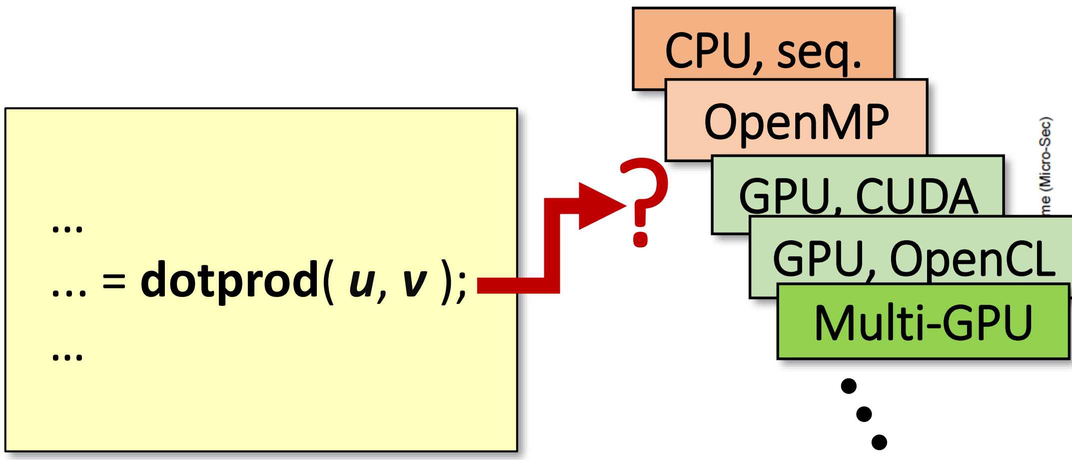
Data-containers for array-based skeleton-call operands

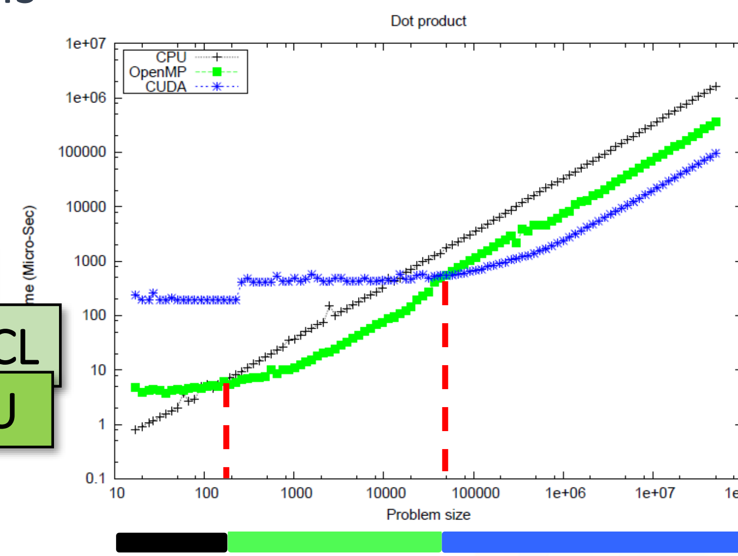
Skeleton instantiation: generates a multi-backend function

Call the skeleton instance

Automatic Backend Selection for a Call

- For a **skeleton-instance call**, find *automatically* the expected **best backend** and settings depending on execution **context** (operand data size / location)
  - Off-line training of performance models
  - Fast lookup at runtime





- Or pre-select a backend manually
  - global default or call-specific

U. Dastgeer, L. Li, C. Kessler: **Adaptive implementation selection in the SkePU skeleton programming library**. Proc. APPT-2013, Stockholm, Sweden, Springer LNCS 8299, pp. 170-183, 2013.

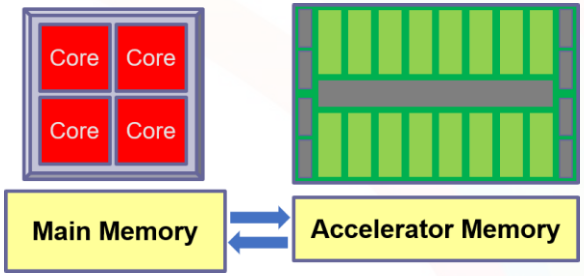
New in SkePU 3: Multi-Variant User Functions

- In ***addition*** to the default (universal) user function, further **platform-specific user function variants** can be provided e.g. by a platform expert programmer.
    - Can also apply to multiple elements at a time
    - SkePU generates *additional* backend variants from these, selectable only if the feature is available on the target system
- Example: Custom SIMDization, special instructions

A. Ernstsson, C. Kessler: **Multi-variant User Functions for Platform-aware Skeleton Programming.** ParCo-2019, Prague, Sep. 2019, IOS press, 2020.

Smart Data-Containers

- Extended generic STL-like containers (**Vector**<>, **Matrix**<>, **Tensor3**<>, **Tensor4**<>)
- Software caching of operand element data in device memories
- Runtime optimization of data transfers, memory management
  - Lazy data transfer [Enmyren, K.'10]
- Iterators
  → Coherence at arbitrary *sub*array level
  [Dastgeer, K. *IJPP*'16]
  - Copy plan – locate closest overlapping valid copies
  - Reuse invalidated allocations of device memory
  - Speedup (up to 10^3 x over 'dumb' containers on iterative applications)



skepu::**Vector<float>** ***a***, ***w***, ***x***;
…
skel1 (***w***, ***a***);
…
skel2 (***x***, ***w***);

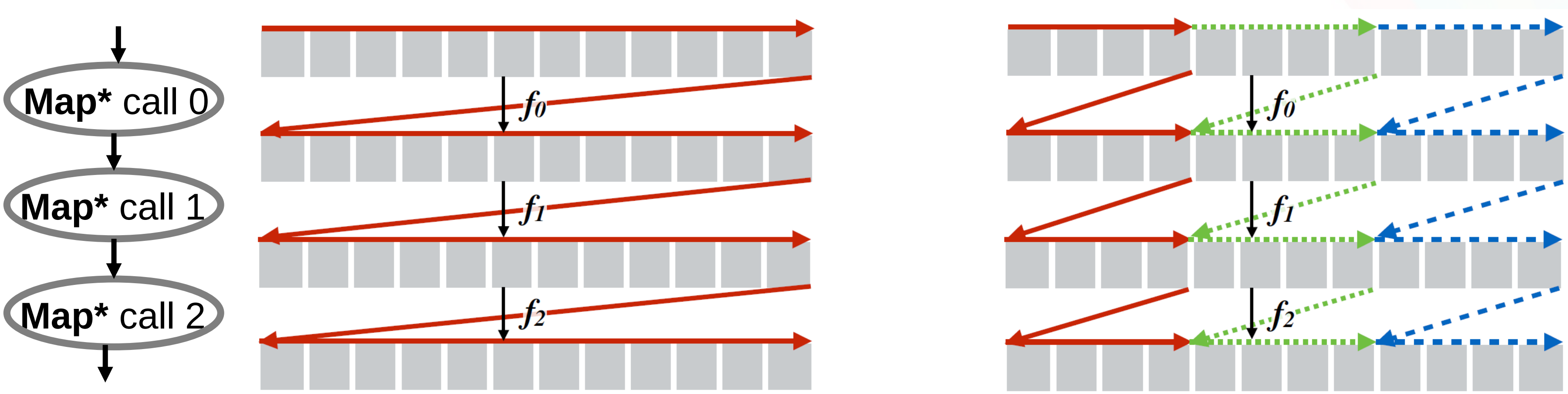
CPU? GPU1? ...
Write ***w***
Read ***w***
CPU? GPU1? ...

U. Dastgeer, C. Kessler: **Smart containers and skeleton programming for GPU-based systems**. *International Journal of Parallel Programming* 44(3):506-530, June 2016

Lazy Execution
of Transformation-Style (*Map, MapOverlap*) Skeleton

- **Lazy execution of Map* calls**: build run-time dependence graph
  - **Lineage** of data-containers – cf. Spark RDDs
  - Not limited to static scope – e.g., entire iterative stencil loop
- Lineage is evaluated when needed (e.g., at a reduction/scan) by **tiling** the postponed **Map*** computations
  - Efficient cache utilization on CPU, local memory on accelerator
  - Tile size is a tuning parameter

Standard (eager) execution **Lazy, tiled execution**



A. Ernstsson, C. Kessler: **Extending smart containers for data locality-aware skeleton programming**. *Concurrency and Computation: Practice and Experience*, 31(5), Mar. 2019, Wiley. DOI: 10.1002/cpe.5003

Cluster Execution Support in SkePU

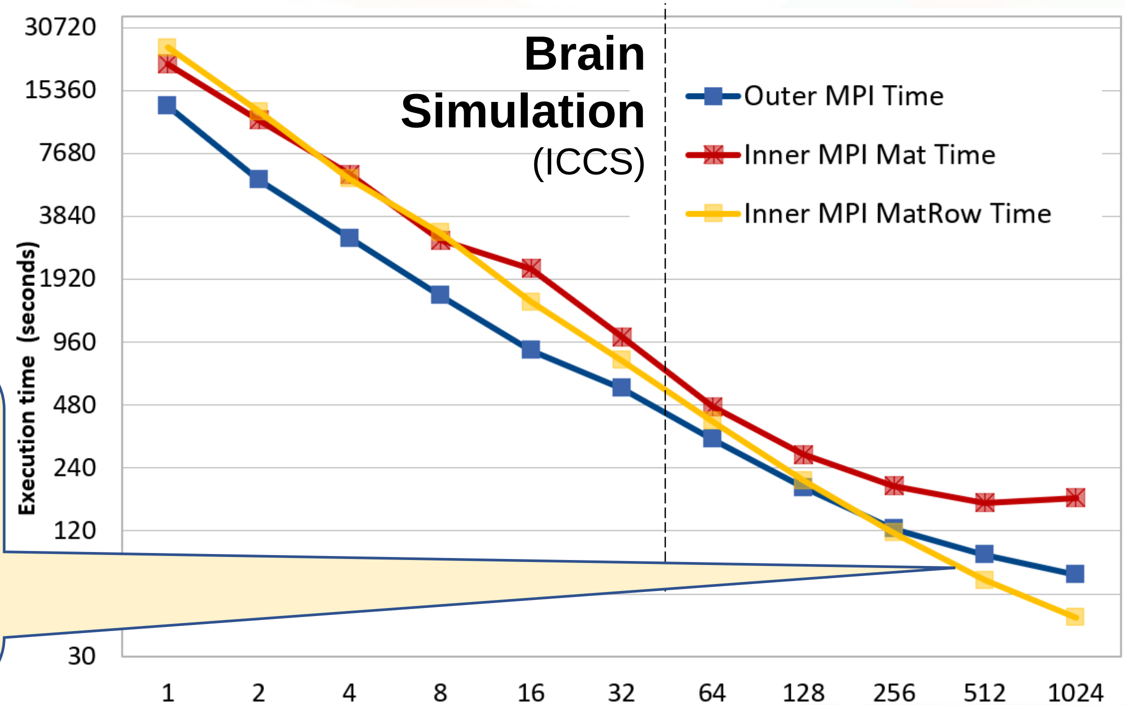
2 “modes” of cluster support in SkePU code:

- ***Outer* MPI** (user-written MPI + SkePU at node level)
- ***Inner* MPI** (SkePU uses MPI internally – cluster backend)

SPMD exec. abstracted away. No syntactic difference from single-node SkePU code

Poor scaling if using the most general matrix access window (**Mat**<>, full random access).

MatRow<> scales much better.



on *Tetralith* cluster at NSC Linköping

A. Ernstsson, J. Ahlqvist, S. Zouzoula, C. Kessler: **SkePU 3: Portable high-level programming of heterogeneous systems and HPC clusters.** Accepted for *Int. J. of Parallel Programming*, 2021

Programmability: One Code to Rule Them All

Sequential C++:

```
...
int main()
{
   double s;
   int i;
   … arrays A, B …
   s = 0.0;
   for (i=0; i<N; i++)
        s += A[i] * B[i];
   ...
}
```

OpenMP:

```
#include <omp.h>
...
int main()
{
   double s;
   … arrays A, B …
  s = 0.0;
#pragma omp parallel shared(s, A, B)
 {
#pragma omp for reduction ( s: + )
   for (i=0; i<N; i++)
        s += A[i] * B[i];
   ...
 }
}
```

MPI:

```
#include <mpi.h>
...
int main()
{
   double s;
   … arrays A, B …
   ... local arrays myA, myB ...
   ... calculate local size myN ...
   MPI_Scatter( A, N, MPI_double, myA,myN,
                   ..., 0, MPI_COMM_WORLD
);
   mys = 0.0;
   for (i=0; i<myN; i++)
        mys += myA[i] * myB[i];
   MPI_Reduce( &mys, 1, MPI_double,
                   &s, 1, MPI_double, 0,
                   MPI_COMM_WORLD);
}
```

Valid C++(11+) code.
No need to write platform-specific code

SkePU source code:

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

float mult ( float a, float b )
{
   return a * b;
}

int main()
{
   … arrays A, B …
   Vector<float> u( A, N );
   Vector<float> v( B, N );
   auto dotprod = MapReduce<2>
   ... = dotprod( u, v );
}
```

Multi-GPU:

OpenCL:

Hybrid:

CUDA:

```
#include <cuda.h>
...

__global__ void dot ( int *a, int *b, int *s)
{
   __shared__ int temp[THREADS_PER_BLOCK]
   int index = threadIdx.x + blockIdx.x * blockDim
   temp[threadIdx.x] = a[index] * b[index];
   __syncthreads();
   if (threadIdx.x == 0) {
      int sum = 0;
      for (int i = 0; i < N; i++) {
         sum += temp[i];
      }
      atomicAdd(s, sum);
   }
}

int main()
{
   ... allocate arrays A, B ...
   int *dev_A, *dev_B
   int siz
   ...
   ... >>>(dev_A, d
   ... dev_s, sizeof(int), cudaMemcp
   ...ree(dev_A);
   cudaFree(dev_B);
   cudaFree(s);
   ...
}
```

No complex code base cluttered with #ifdefs for various platform-specific code versions

SkePU Evolution Milestones

	SkePU 1 (2010)	SkePU 2 (2016)	SkePU 3 (2020)
API based on	C, C++ (pre-2011), C preprocessor	C++11, Precompiler (clang)	C++11, Precompiler
Skeletons	Map, Reduce, Scan, MapReduce, MapArray, MapOverlap, Generate	Map, Reduce, Scan, MapReduce, MapOverlap, Call	Map, Reduce, MapReduce, MapPairs/MapPairsReduce, Scan, MapOverlap$^{\text{NEW}}$, Call ...
User functions as	C preprocessor macros Not type-safe	C++ functions Multi-way variadic Type-safe	As SkePU 2, + multi-variant user func.s + multi-value return
Data-containers, Access proxy	Vector<>, Matrix<>	Vector<>, Matrix<>	Vector<>, Matrix<>, Tensor3<>, Tensor4<>; MatRow<>, Region*ND*<>, ...
Platforms supported	CPU (C, OpenMP), GPU (CUDA, OpenCL); Myriad	CPU (C++, OpenMP), GPU (CUDA, OpenCL)	CPU, GPU, hybrid CPU/GPU, Cluster (StarPU-MPI)
Memory model	Sequential consistency	Seq. consistency	Weak consistency (default), alternatively sequential c.

Summary, so far

Goal: **Simplify** the use of heterogeneous parallel systems
and enable writing **future-proof, portable, high-level** code

- Skeletons implement parallelizable patterns
- Single-source, sequential-looking code
- **SkePU** skeleton programming framework
  - API: Multi-variadic, customizations, type-safe, compatibility with C++11
  - Smart data-containers
  - Data-container access windows model access patterns
  - Multi-backend, tunable selection
- Open source, https://skepu.github.io
- Used in research and teaching
- Lots of ideas for the future
- Cooperations welcome!

Now let's take a closer look at SkePU ...

Thanks!

skepu.github.io

Glossary

- **Performance Portability**

  … is the ability of a program to automatically adapt to a new execution platform to achieve an automated best-effort optimization of performance on the new target system, without manual rewriting / reoptimization.

- **[Algorithmic] Skeleton**

  … is a pre-defined, generic software construct
  for high-level programming
  that implements a specific *pattern* of control and data flow,
  that can be *parameterized* by problem-specific code
  to instantiate a problem-specific function, and
  whose implementation internally *encapsulates*
  all platform-specific details such as parallelism,
  heterogeneity, communication and synchronization.

References on SkePU

Johan Enmyren, Christoph Kessler: **SkePU: A multi-backend skeleton programming library for multi-GPU systems.** Proc. 4th Int. Workshop on High-Level Parallel Programming and Applications (HLPP-2010), Baltimore, USA, Sep. 2010. ACM.

Usman Dastgeer, Johan Enmyren, Christoph Kessler: **Auto-tuning SkePU: A multi-backend skeleton programming framework for multi-GPU systems.** Proc. IWMSE-2011, Hawaii, USA, May 2011, ACM.

Usman Dastgeer, Christoph Kessler. **Flexible runtime support for efficient skeleton programming on hybrid systems**. ParCo'11: Int. Conf. on Parallel Computing. Ghent, Belgium, 2011.

Usman Dastgeer, Lu Li, Christoph Kessler: **Adaptive implementation selection in the SkePU skeleton programming library**. Proc. Biennial Conf. on Advanced Parallel Processing Technology (APPT-2013), Stockholm, Sweden, Aug. 2013. Springer LNCS 8299, pp. 170-183, 2013.

Usman Dastgeer, Christoph Kessler: **Smart containers and skeleton programming for GPU-based systems**. *Int. Journal of Parallel Programming*, March 2015. DOI: 10.1007/s10766-015-0357-6

August Ernstsson, Lu Li, Christoph Kessler: **SkePU 2: Flexible and type-safe skeleton programming for heterogeneous parallel systems**. *International Journal of Parallel Programming* 46(1):62, Jan. 2018

August Ernstsson, Christoph Kessler: **Extending smart containers for data locality-aware skeleton programming**. *Concurrency and Computation: Practice and Experience*, 31(5), Mar. 2019, Wiley. DOI: 10.1002/cpe.5003

Tomas Öhberg, August Ernstsson, Christoph Kessler: **Hybrid CPU-GPU execution support in the skeleton programming framework SkePU**. *The Journal of Supercomputing*, Springer. July 2020. DOI: 10.1007/s11227-019-02824-7

August Ernstsson, Christoph Kessler: **Multi-variant user functions for platform-aware skeleton programming.** ParCo-2019 conference, Prague, Sep. 2019, in: I. Foster et al. (Eds.), *Parallel Computing: Technology Trends*, series: *Advances in Parallel Computing*, vol. 36, IOS press, March 2020, pages 475-484. DOI: 10.3233/APC200074.

August Ernstsson: **Designing a Modern Skeleton Programming Framework for Parallel and Heterogeneous Systems**. Licentiate Thesis no. 1886, Nov. 2020, Linköping University. DOI: 10.3384/lic.diva-170194 https://doi.org/10.3384/lic.diva-170194

August Ernstsson, Johan Ahlqvist, Stavroula Zouzoula, Christoph Kessler: **SkePU 3: Portable high-level programming of heterogeneous systems and HPC clusters.** *Int. J. of Parallel Programming,* May 2021. https://link.springer.com/article/10.1007/s10766-021-00704-3

Lazaros Papadopoulos, Dimitrios Soudris, Christoph Kessler, August Ernstsson, Johan Ahlqvist, Nikos Vasilas, Athanasios I. Papadopoulos, Panos Seferlis, Charles Prouveur, Matthieu Haefele, Samuel Thibault, Athanasios Salamanis, Theodoros Ioakimidis, Dionysios Kehagias: **EXA2PRO: A Framework for High Development Productivity on Heterogeneous Computing Systems** Accepted for *IEEE Transactions on Parallel and Distributed Systems*, 2021.

SkePU documentation and download: https://skepu.github.io