A Framework for Performance-aware Composition of Applications for GPU-based Systems

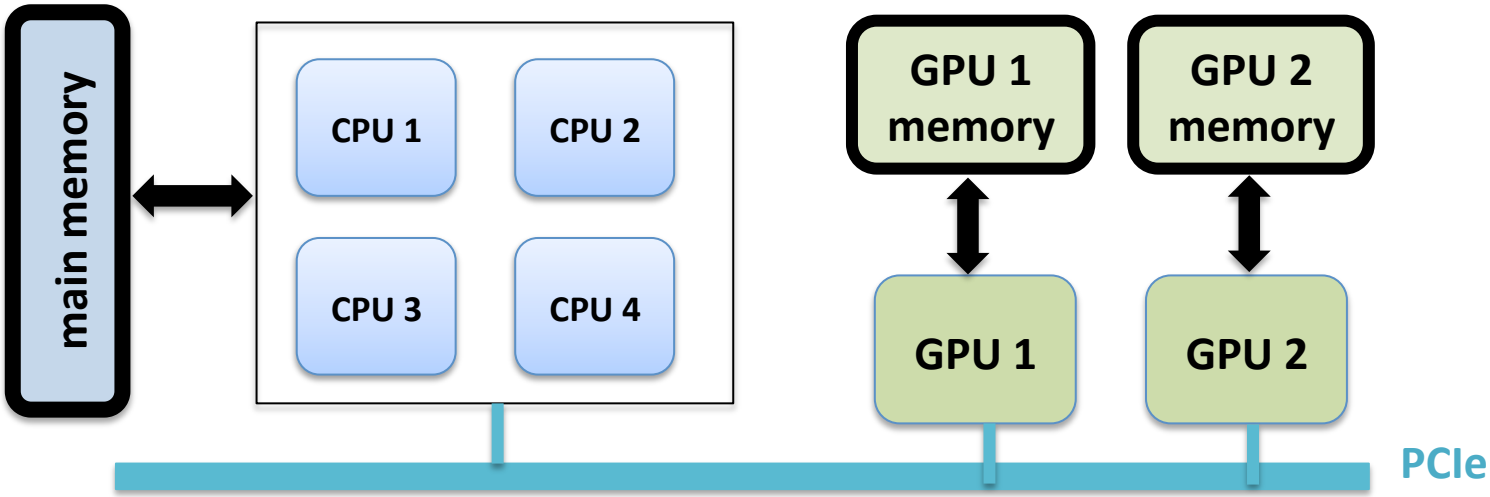
Usman Dastgeer, Christoph Kessler

Short title

Optimized component composition for GPU based systems.

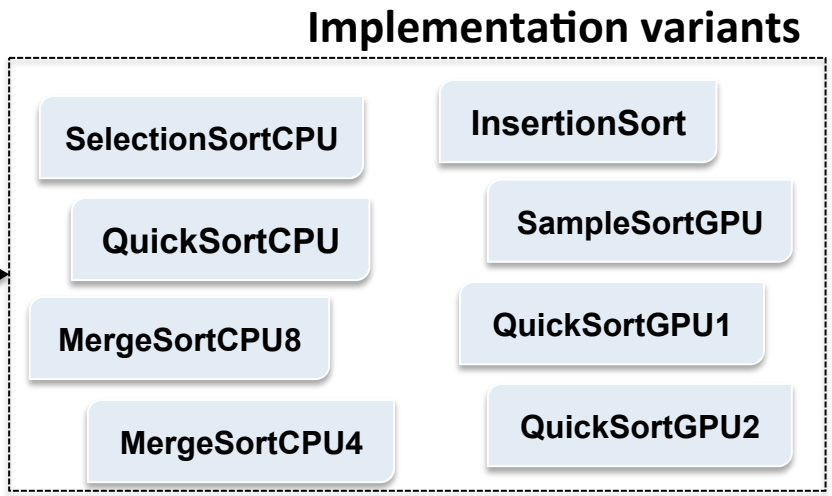
Optimized component composition for GPU-based systems.

A system containing multi-core CPUs and one or more programmable GPUs (GPGPU computing)

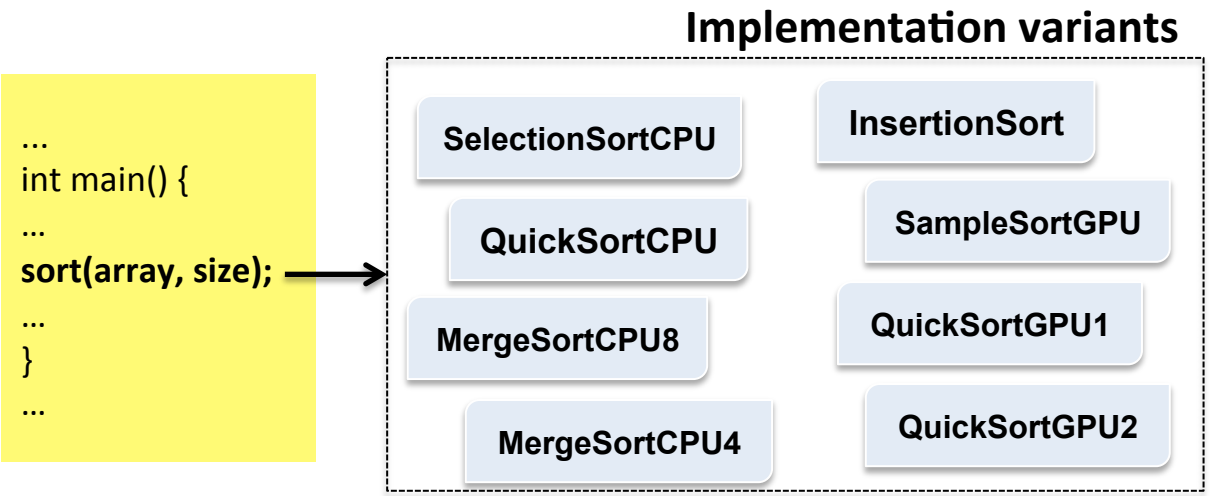


Optimized component composition for GPU-based systems.

```
...
int main() {
...
sort(array, size);
...
}
...
```

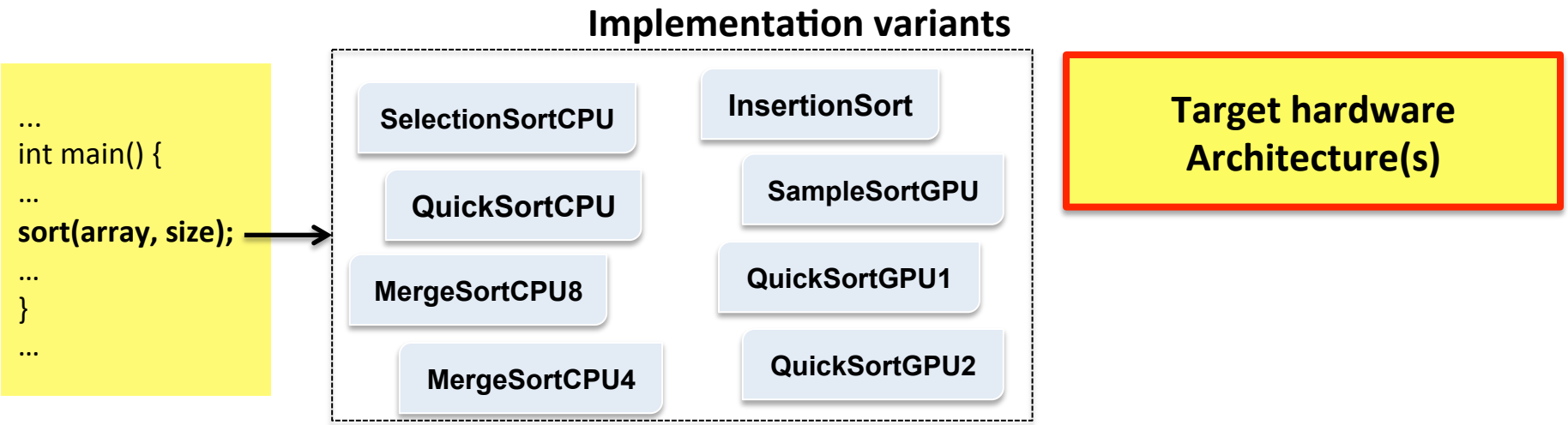


Optimized component composition for GPU-based systems.



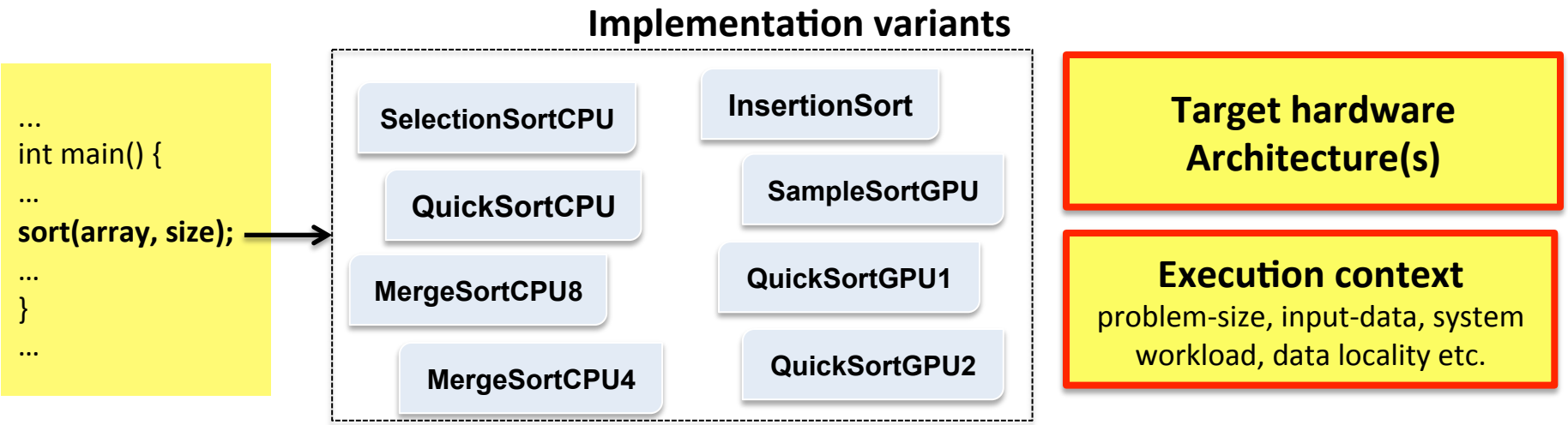
Which one to use?

Optimized component composition for GPU-based systems.



Which one to use?

Optimized component composition for GPU-based systems.



Which one to use?

Optimized component composition for GPU-based systems.

```
...
int main() {
...
sort(array, size);
...
}
...
```

Implementation variants

- SelectionSortCPU
- QuickSortCPU
- MergeSortCPU8
- MergeSortCPU4
- InsertionSort
- SampleSortGPU
- QuickSortGPU1
- QuickSortGPU2

Target hardware Architecture(s)

Execution context
problem-size, input-data, system workload, data locality etc.

Which one to use?

Solution should be:

- Generic
- Scalable
- Portable

Our Focus

1. Support currently established programming models (OpenMP, Pthreads, CUDA, OpenCL etc.)
   - Different from PetaBricks, EXOCHI, Merge, IBM's Liquid metal, Elastic computing etc.
   - Easy to use/wrap existing library/legacy code.
2. Support for staged composition
   - Selection decisions made statically and/or dynamically
3. Support *different composition needs*

Performance-aware composition

Different composition needs of applications

- Implementation selection ➔ CPU **or** GPU
  - Models calibrated offline
  - Models learned online
- Hybrid Execution ➔ CPU **and** GPU
- Global composition
  - >1 component call, make better decisions by:
    - Lookahead,
    - Control and data-flow analysis

!!! A single framework to support all above !!!

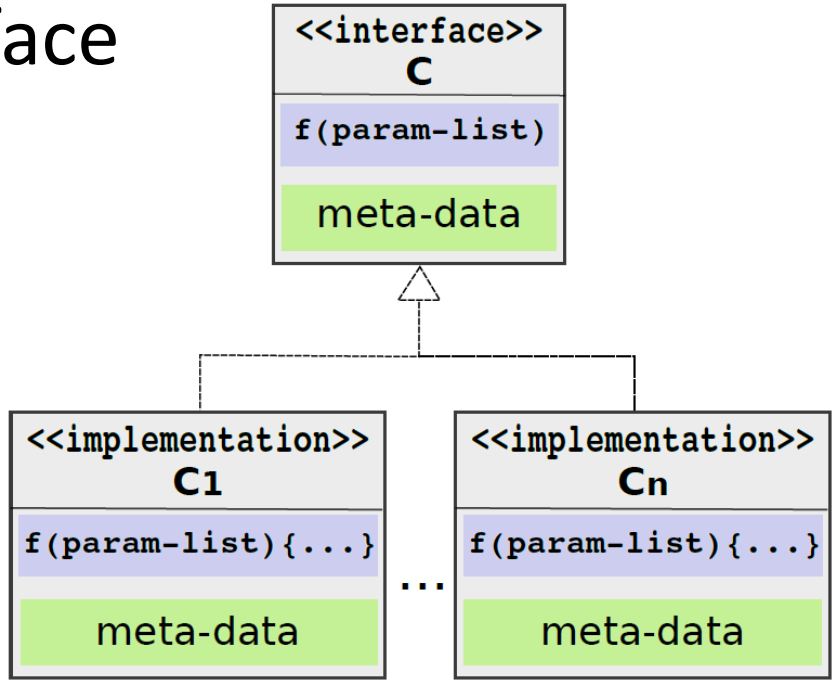
In this work

We have developed a:

1. Component model
2. Global Composition Framework (GCF)

Component model

- Notion of interface and multiple implementations (or variants) of that interface



Component model

- Notion of interface and multiple implementations (or variants) of that interface
- Pragma annotations to specify meta-data:
  - Interface meta-data
    - interface/function name, access-pattern/size for parameters, performance model etc.

Component model

- Notion of interface and multiple implementations (or variants) of that interface
- Pragma annotations to specify meta-data:
  - Interface meta-data
  - Implementation meta-data
    - interface it implements, type (openmp, cuda etc.), target (CPU, GPU), performance model etc.

Component model

- Notion of interface and multiple implementations (or variants) of that interface
- Pragma annotations to specify meta-data:
  - Interface meta-data
  - Implementation meta-data
  - Call meta-data
    - interface name, constraints on call etc.

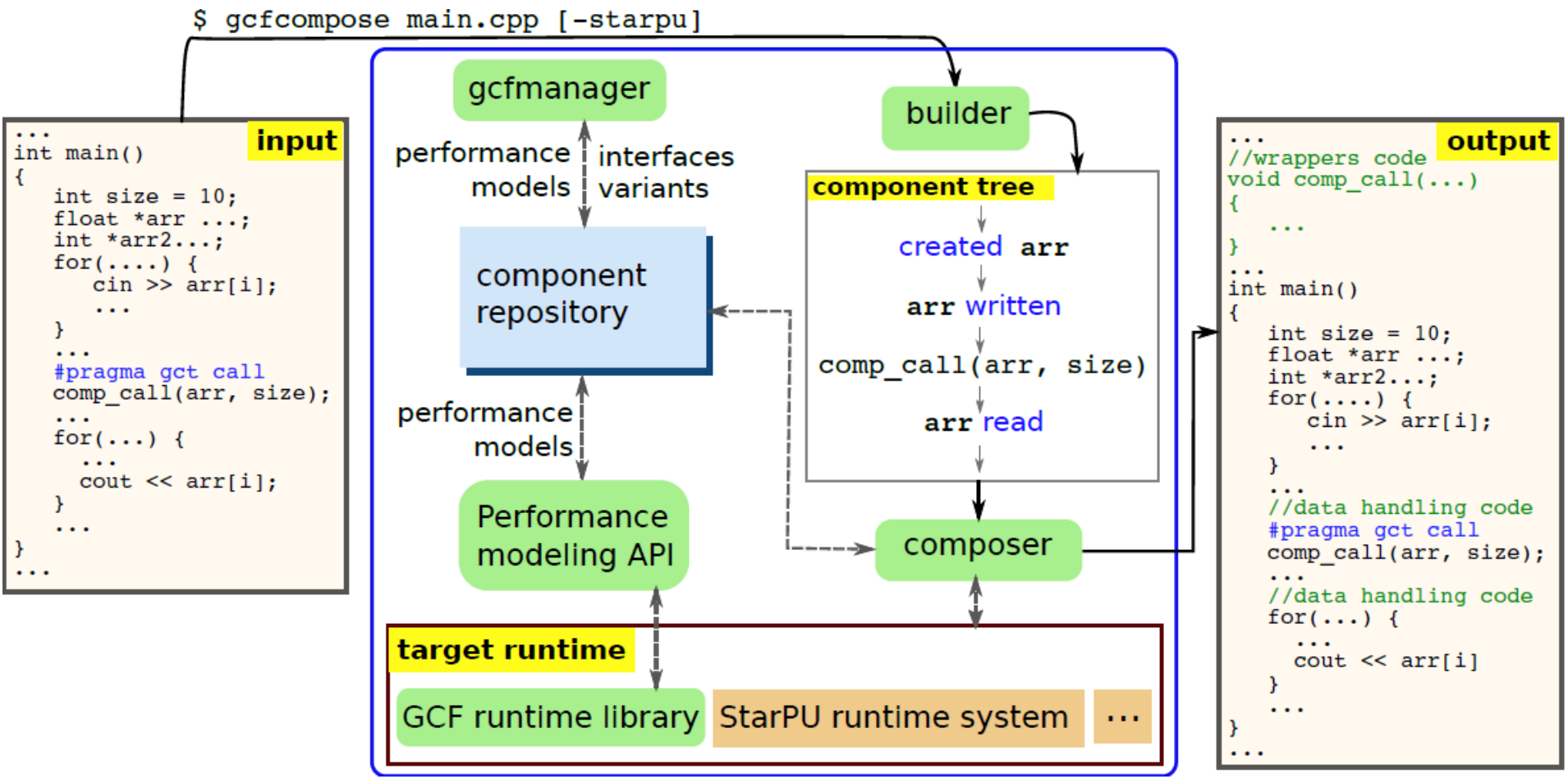
Component model

- Notion of interface and multiple implementations (or variants) of that interface
- Pragma annotations to specify meta-data:
  - Interface meta-data
  - Implementation meta-data
  - Call meta-data
  - Data access hints
    - access type for data operands...

Global composition framework

- Based on ROSE source-to-source compiler
  - Supports program analysis (control, data, AST etc.)
  - Can modify, insert source code at different program locations.
  - Can support more complex composition scenarios

Global composition framework



Global composition framework

- Five parts beside component model
  1. A component repository manager (gcfmanager)
  2. Component tree builder
  3. Composer
  4. Performance modeling API
  5. Runtime library

Global composition framework

1 **Component repository manager (gcfmanager)**

- Command-line utility
- (un-) registering components (interfaces, variants, models)
- Manage existing (registered) components

2 **Component tree builder**

- Parses source-code AST
- Analyse component calls, operand data flow/accesses
- Build component tree

Global composition framework

3 **Composer**

- **Input:** Component tree + Control flow graph
- Generates modified/new source code for composition
- Can generate code for different runtime systems
  - StarPU
  - (Light-weight) Runtime library
  - More?

Global composition framework

4 **Performance modeling API**

- Generic, flexible, extensible API to model different kinds of performance models
- Can use both empirical, analytical models.
- Can be (partially) calibrated or learned online
- Integration with ”our runtime library”

Global composition framework

4 **Performance modeling API**

– Currently 2 categories:

- **Per-variant** performance-model
  - Each implementation variant has its own model
  - Returns ”estimated cost/time” value for a given context
  - History-based, analytical etc.
- **Per-component** performance-model
  - 1 common model for all variants of a component
  - Internally decide which implementation variant to use
  - Returns ”best implementation variant” name for a given context

Global composition framework

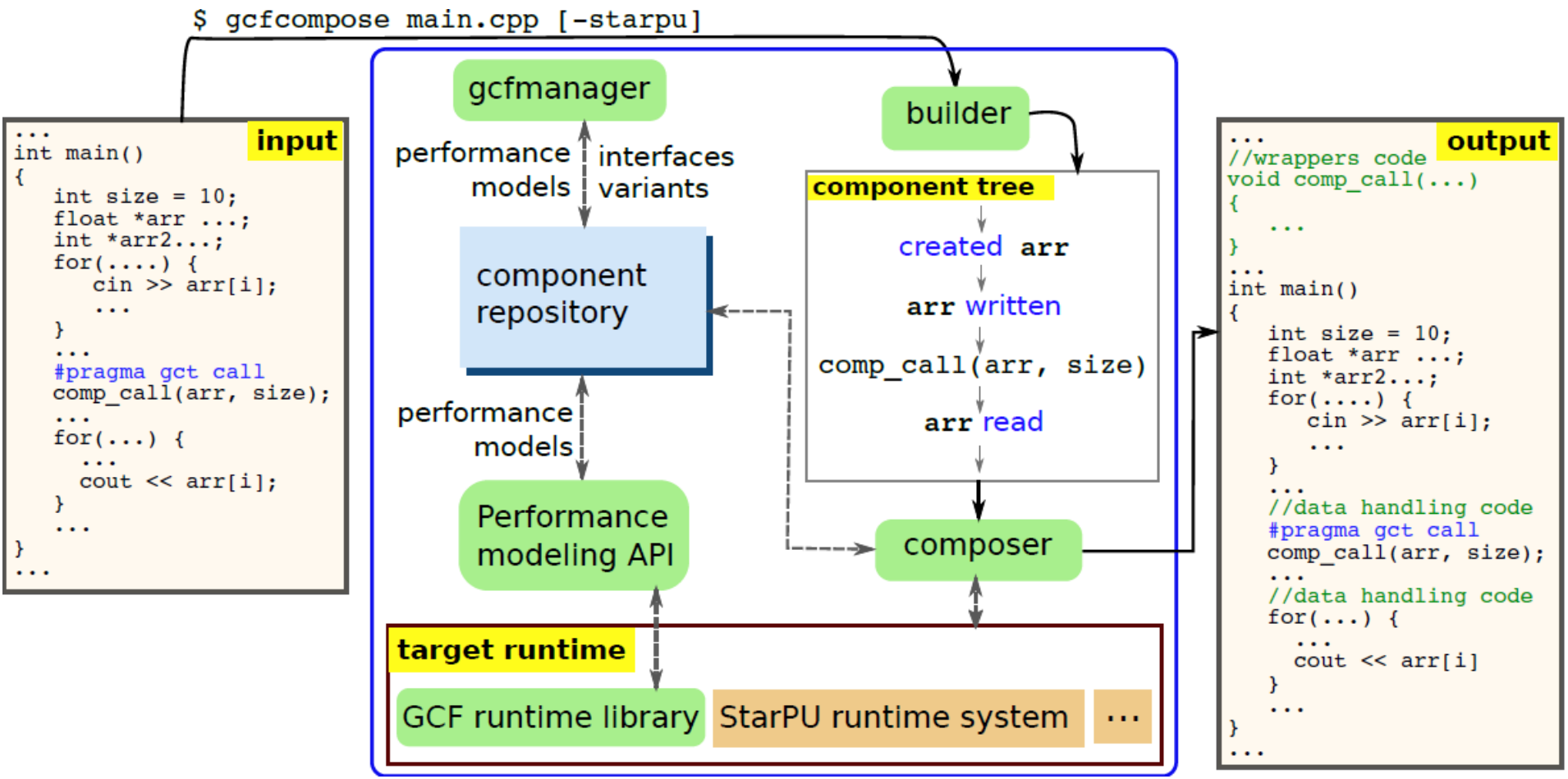
```
class PerfModel
{
  /* for per variant performance models */
  virtual double predict(int keys[], int numKeys);
  virtual void addEntry(int keys[], int numKeys,
                                    double timeVal);

  /* for per interface performance models */
  virtual string predict(int keys[], int numKeys,
                                    double *commCost);
  virtual void addEntry(int keys[], int numKeys,
                        string impl, double timeVal);

  /* common for both kinds */
  virtual void close();
};
```

Global composition framework

5 **Runtime library**

- Focus: light-weight, flexible and extensible
- Initially as **offloading** target
- Focus on OpenMP instead of multiple CPU workers...
- **Current-work:** parallel execution using OpenMP and CUDA workers
- Integrated with performance modeling API discussed earlier.

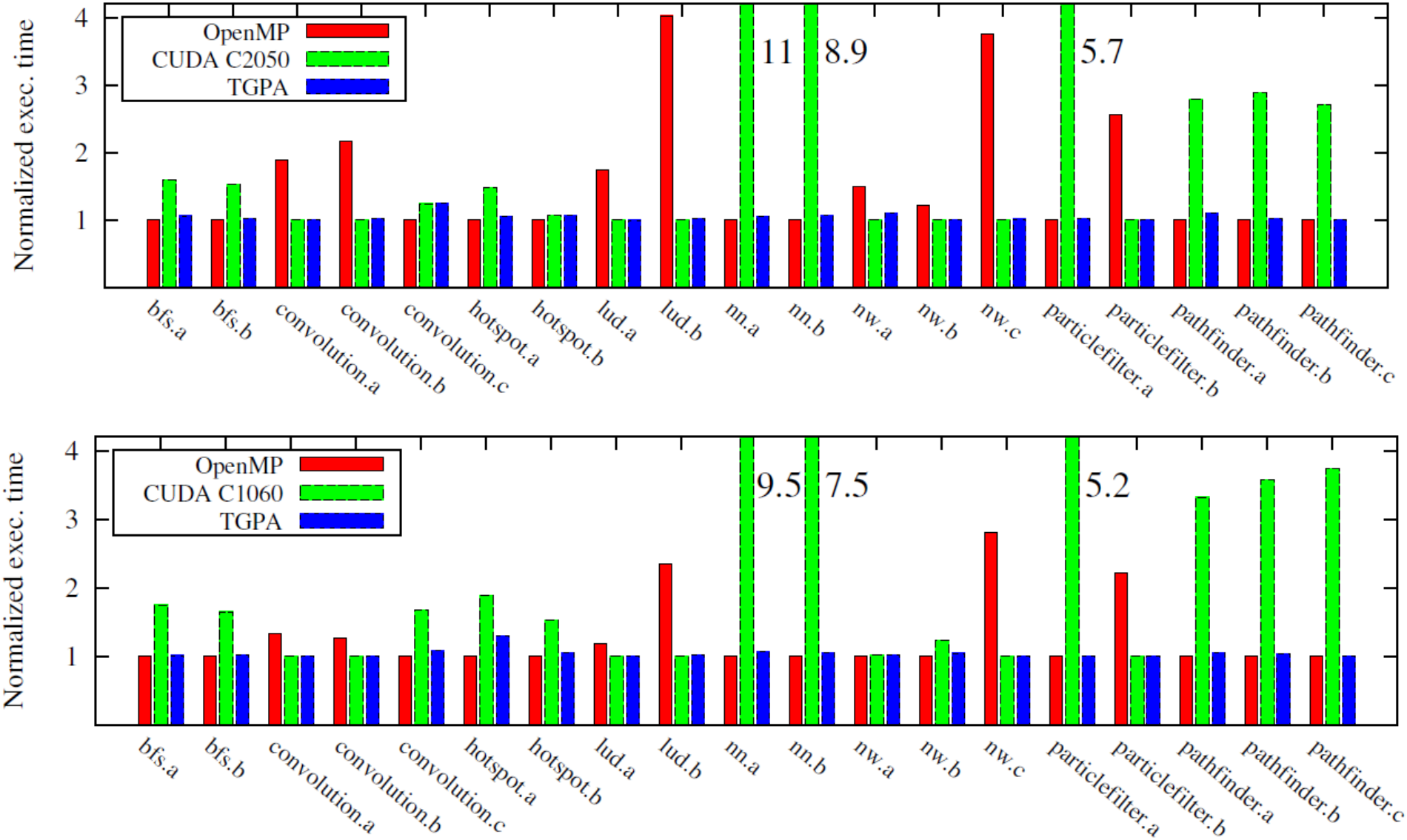
Global composition framework



Evaluation

- Several applications from RODINIA benchmark, scientific kernels, image processing applications
- Two evaluation platforms:
  - **System A:** 2 Xeon E5520 CPUs, NVIDIA C2050 GPU with L1/L2 cache support.
  - **System B:** Xeon X5550 CPUs, NVIDIA C1060 GPU

Evaluation (Impl. Selection) - Offline



Evaluation (Impl. Selection) - Online

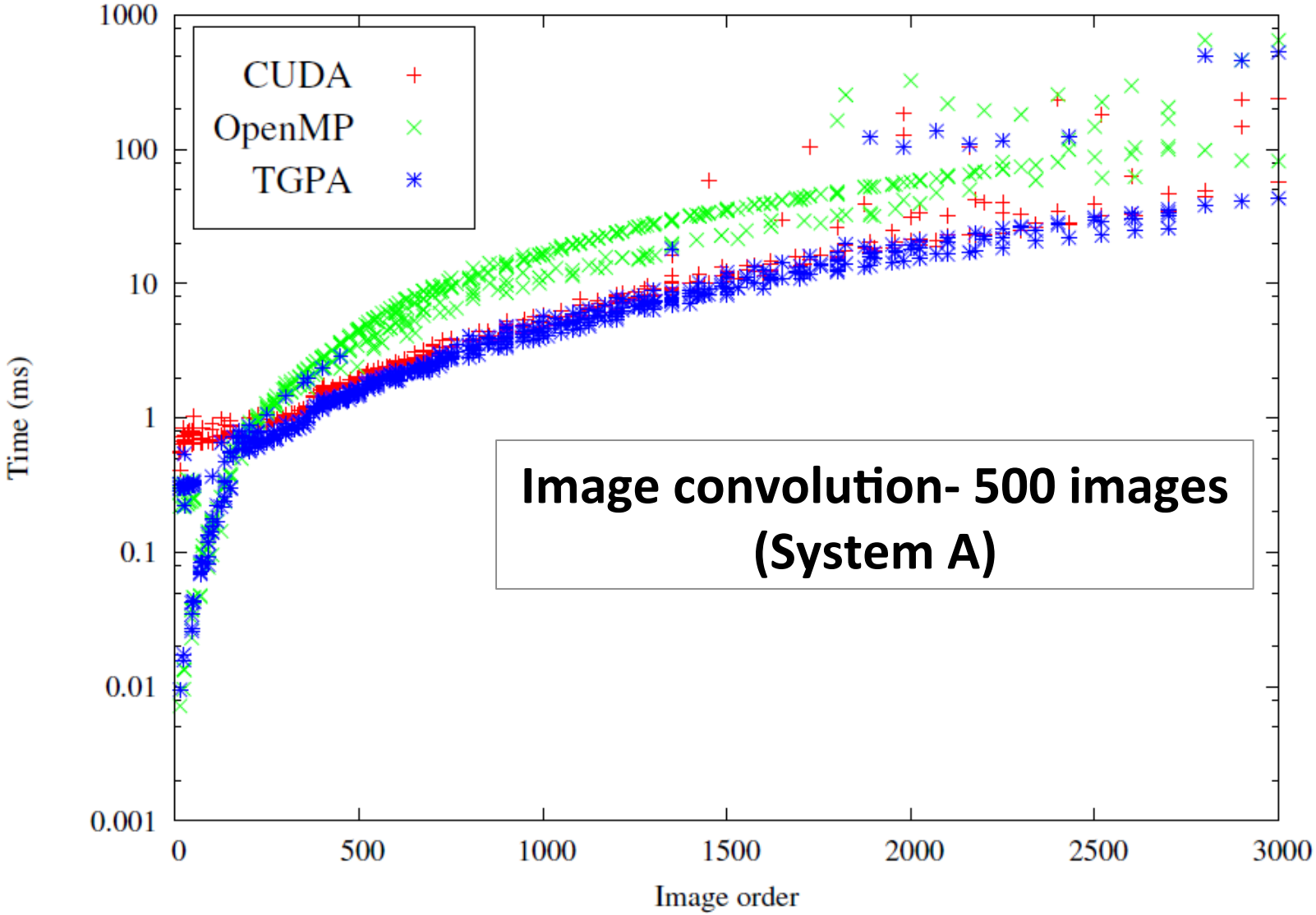
Sorting (2000 files)

	System A		System B	
	Time(ms)	Rel. Time	Time (ms)	Rel. Time
Quick sort CPU	3444.96	3.45	7271.36	4.20
Quick sort CUDA	1134.25	1.20	1961.77	1.13
TGPA	997.24	1.0	1732.73	1.0

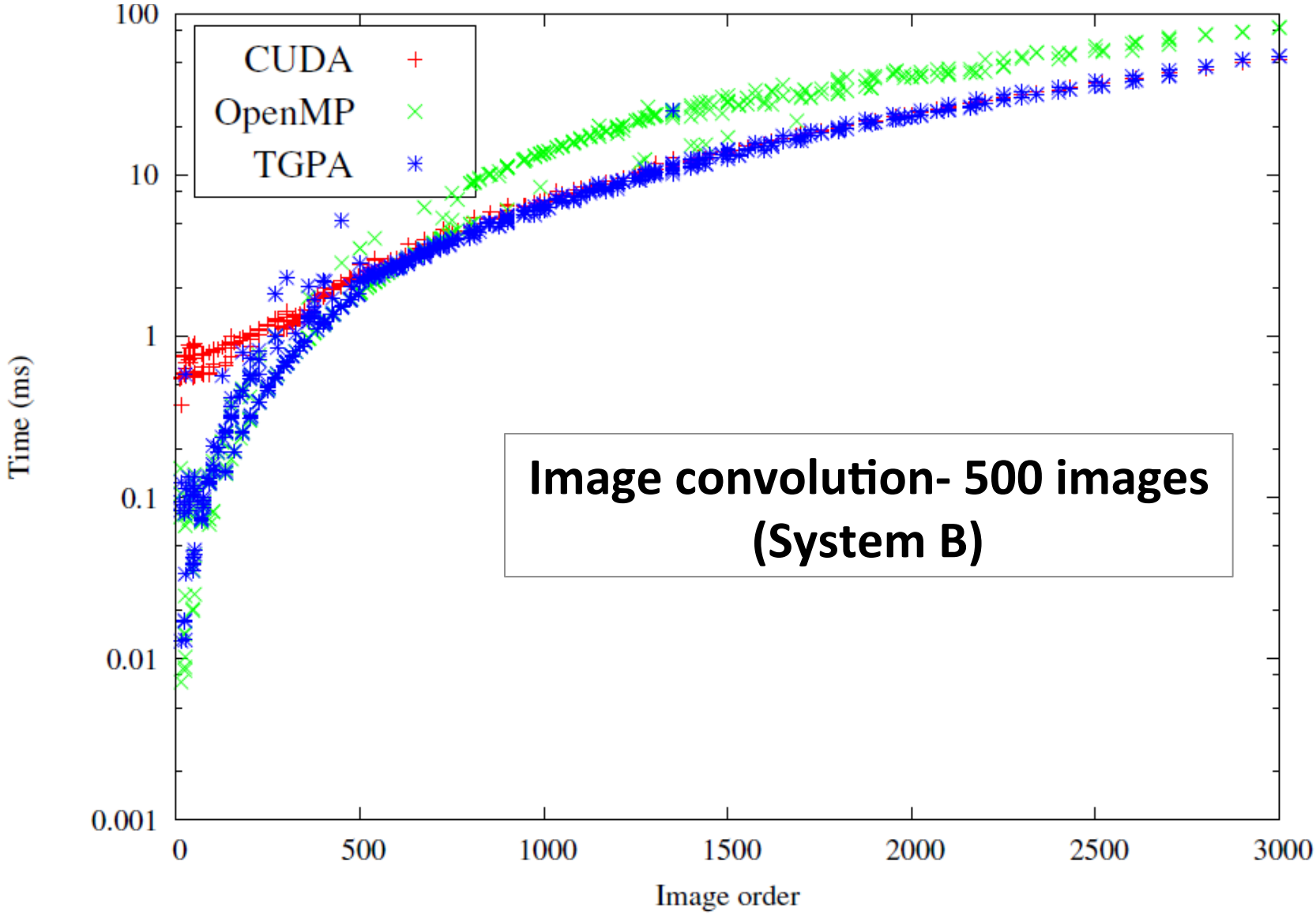
Image convolution (500 images)

	System A		System B	
	Time(ms)	Rel. Time	Time (ms)	Rel. Time
Convolution OpenMP	12276.79	2.75	6951.62	1.74
Convolution CUDA	4997.54	1.12	4598.67	1.15
TGPA	4467.44	1.0	3990.18	1.0

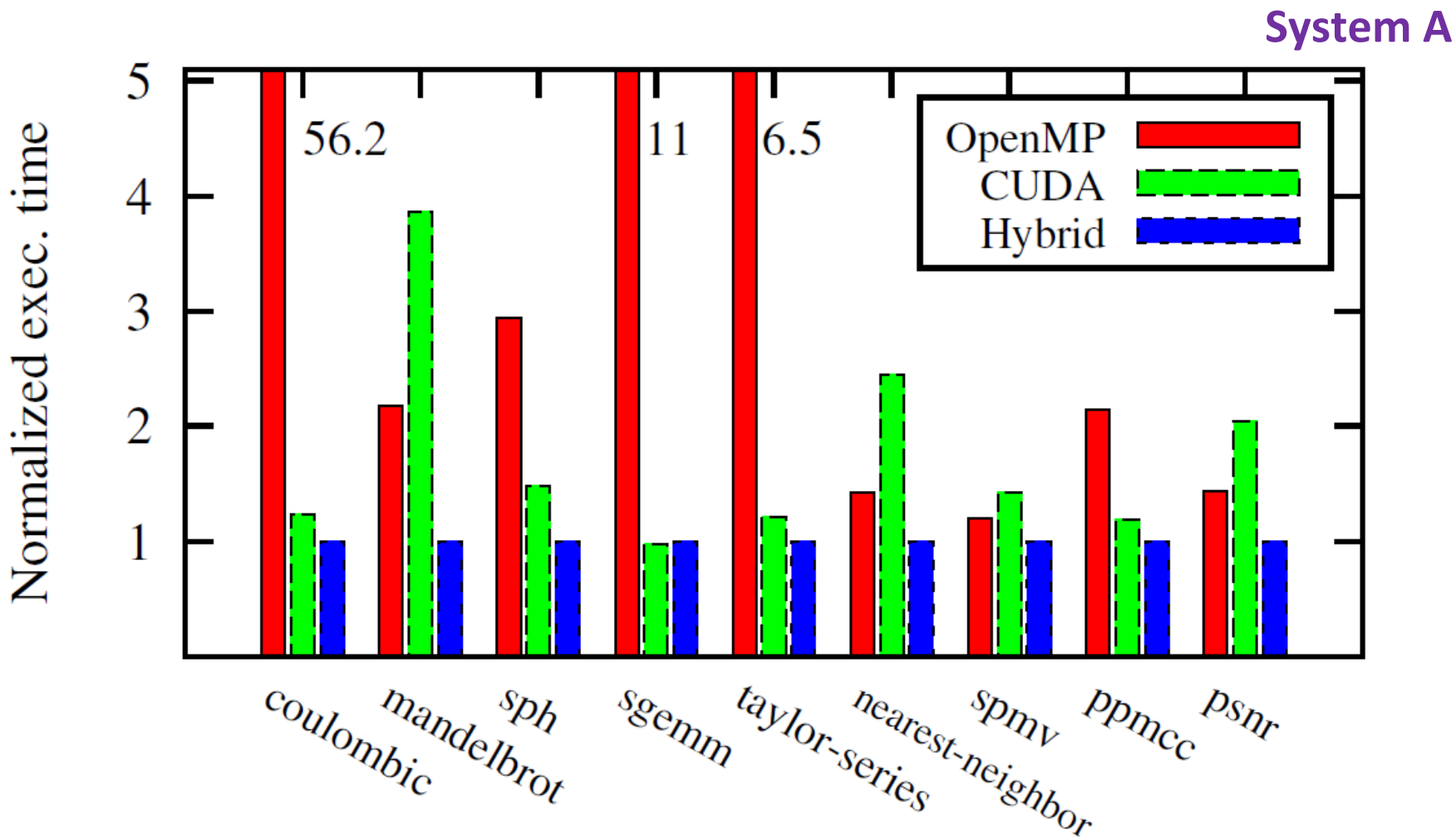
Evaluation (Impl. Selection) - Online



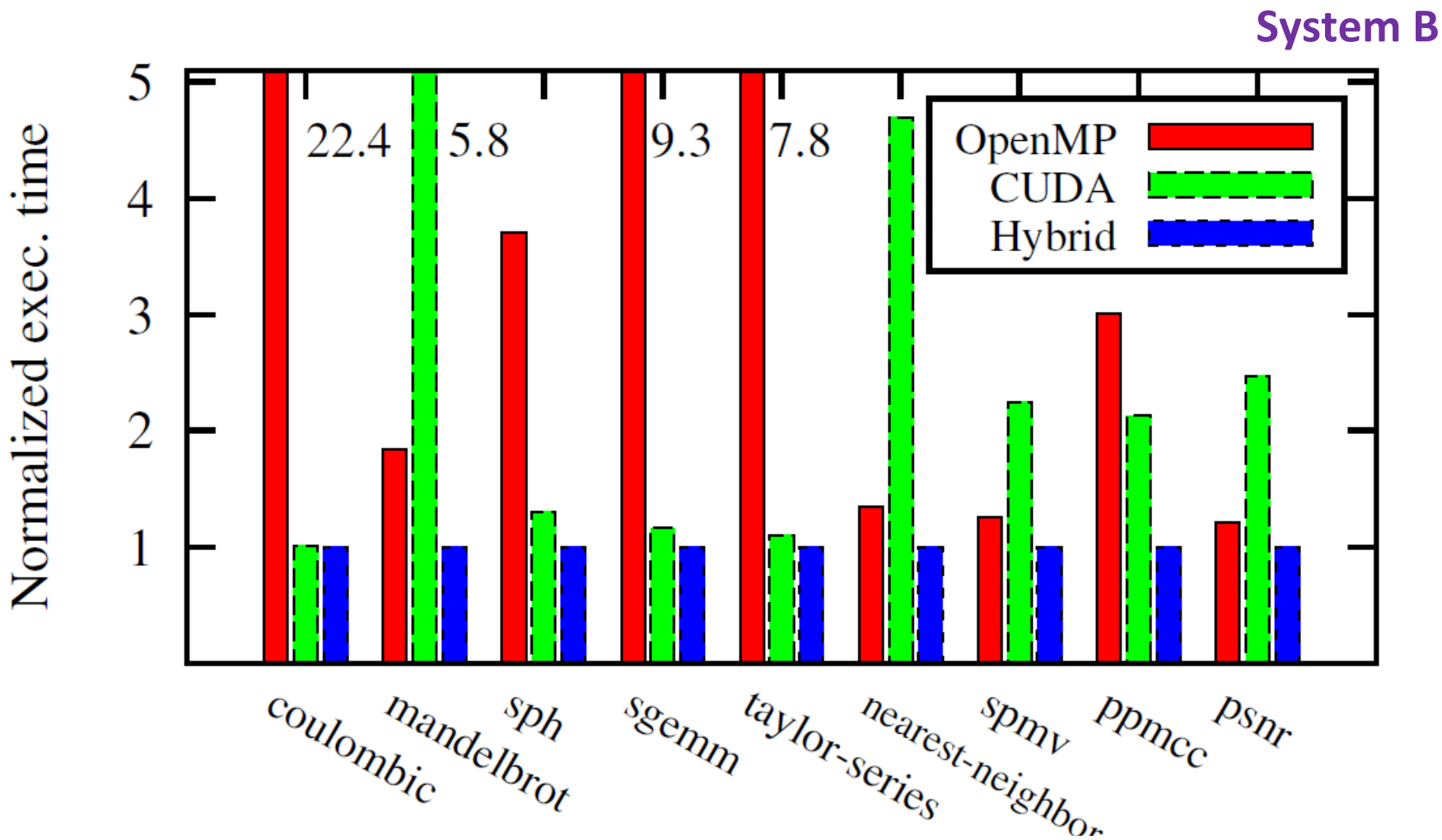
Evaluation (Impl. Selection) - Online



Evaluation (Hybrid execution)



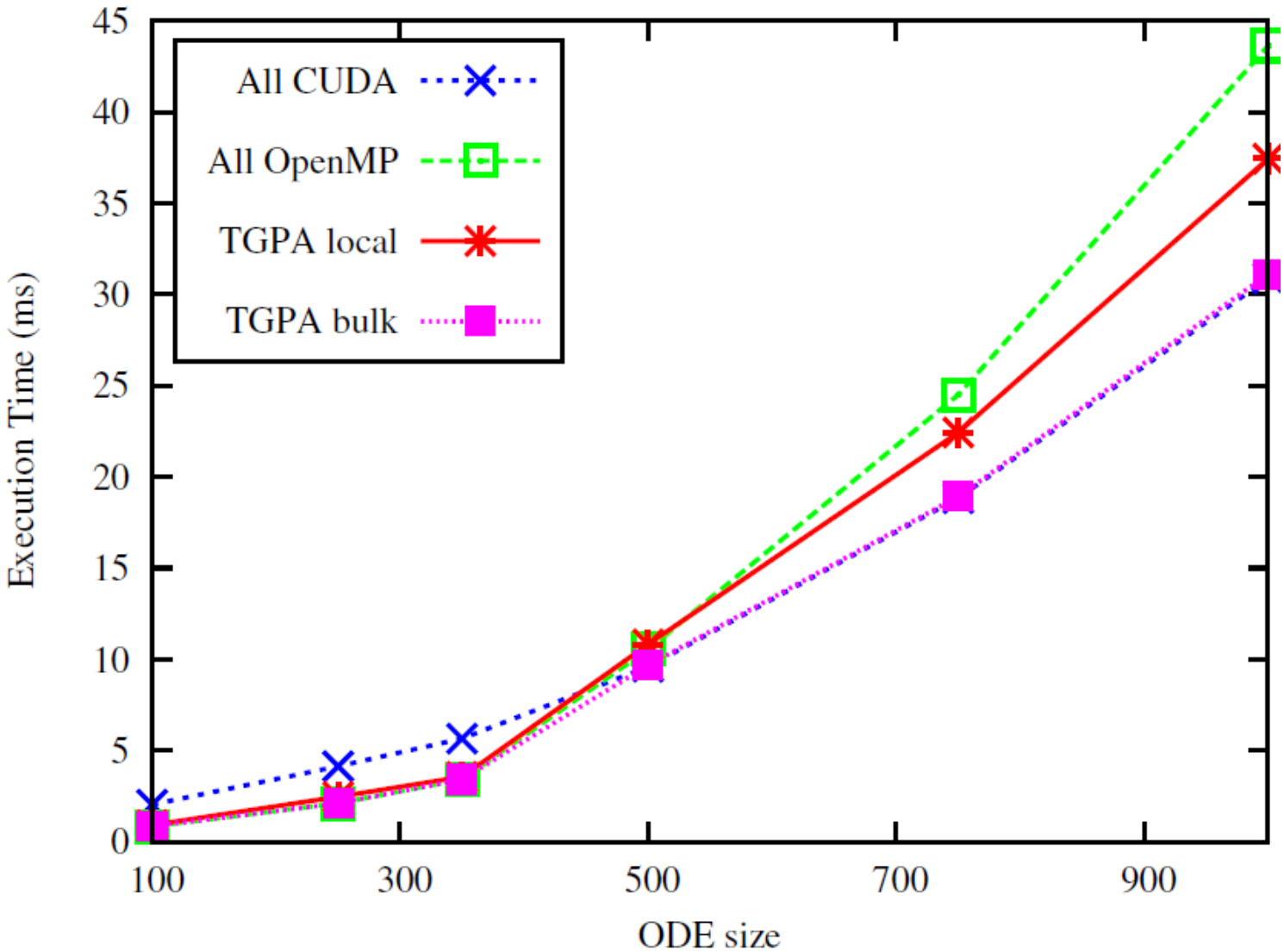
Evaluation (Hybrid execution)



Evaluation (Bulk Composition)

```
// 1D arrays: y, help1, yscal, temp, err_vector
#pragma gct call interface(libsolve_copy)
libsolve_copy(y, help1, ode_size);
...
#pragma gct call interface(absaxpy)
absaxpy(y, help1, yscal, h, ode_size);
...
#pragma gct call interface(axpy)
axpy(help1, y, temp, h, ode_size);
...
#pragma gct call interface(libsolve_copy)
libsolve_copy(temp, y, ode_size);
...
#pragma gct call interface(scale)
scale(help1, err_vector, h, ode_size);
...
#pragma gct call interface(absquotMax)
absquotMax(err_vector, yscal, &error_max, ode_size);
```

Evaluation (Bulk Composition)



Conclusion

- A composition framework for tackling different component composition scenarios.
  - Implementation selection with pre-calibrated/ online-calibrated performance models.
  - Hybrid execution
  - Bulk composition
- **Future work:** Consider more complex 'global' composition scenarios.

Thanks