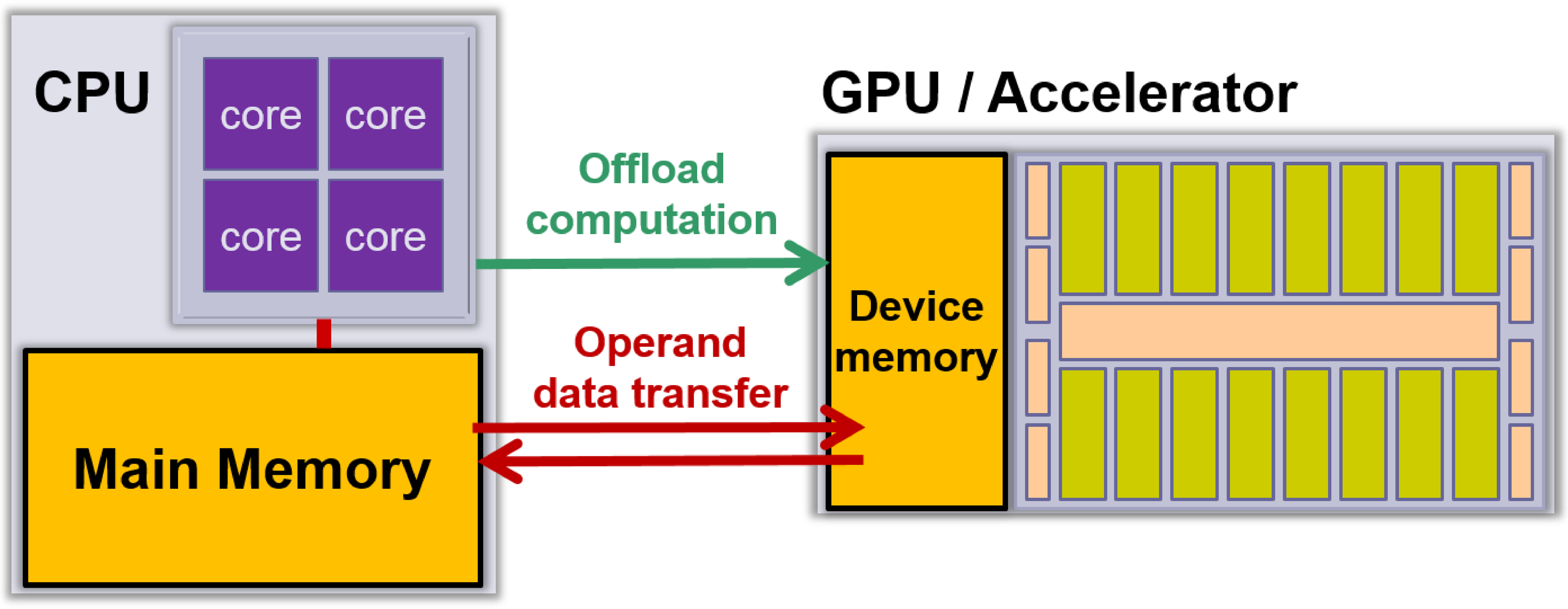
# VectorPU: A Generic and Efficient Data Container

## Enabling Transparent Data Transfer on GPU-based Systems

Lu Li and Christoph Kessler

IDA, Linköping University

## Overview




- Large percentage of code written for data management (82% for a simple example)
- Nvidia's Unified Memory incurs significant overhead.
- We propose VectorPU
  - A high level and efficient data abstraction
  - Enable a unified memory view with STL-like interface
  - Very low overhead
  - Additional optimizations: lazy allocation, optimal transfer fusion

## VectorPU

- C++ template run-time library.
- Expressive annotations but no compiler support required
- Portable to different heterogeneous architectures.
- Significant speedup compared to Nvidia's unified memory
- No noticeable slowdown compared to manually written code

## Annotation of Operands for Access Modes

- R: CPU read, GR: GPU read
- W: CPU write, GW: GPU write
- RW: CPU read and write, GR: GPU read and write
- I: iterator, e.g., RI refers to a CPU read iterator, REI refers to a CPU read end iterator
- ...

## Flow Signature

- Function invocation annotation (one-time)
  - $\alpha$ signature: `foo ( R(x) , W(y), RW(z), size ) ;`
- Function definition annotations (reusable)
  - $\beta$ signature: `#define func_flow (GR)(GW)(GRW)(NA)`
  - $\gamma$ signature: `__global__ void bar(const float *x[[GR]], float *y[[GW]], float *z[[GRW]], int size)`

## Example using Iterator

```
vectorpu::vector<My_Type> x(N);
std::generate(WI(x), WEI(x), RandomNumber);
thrust::sort(GRWI(x), GRWEI(x));
std::copy(RI(x), RI(x), ostream_iterator<My_Type>(cout, ""));
```

## References

[1] L. Li and C. Kessler, "VectorPU: A Generic and Efficient Data-container and Component Model for Transparent Data Transfer on GPU-based Heterogeneous Systems.," in *Proc. 8th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and 6th Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms (PARMA-DITAM'17)*, ACM, 2017.

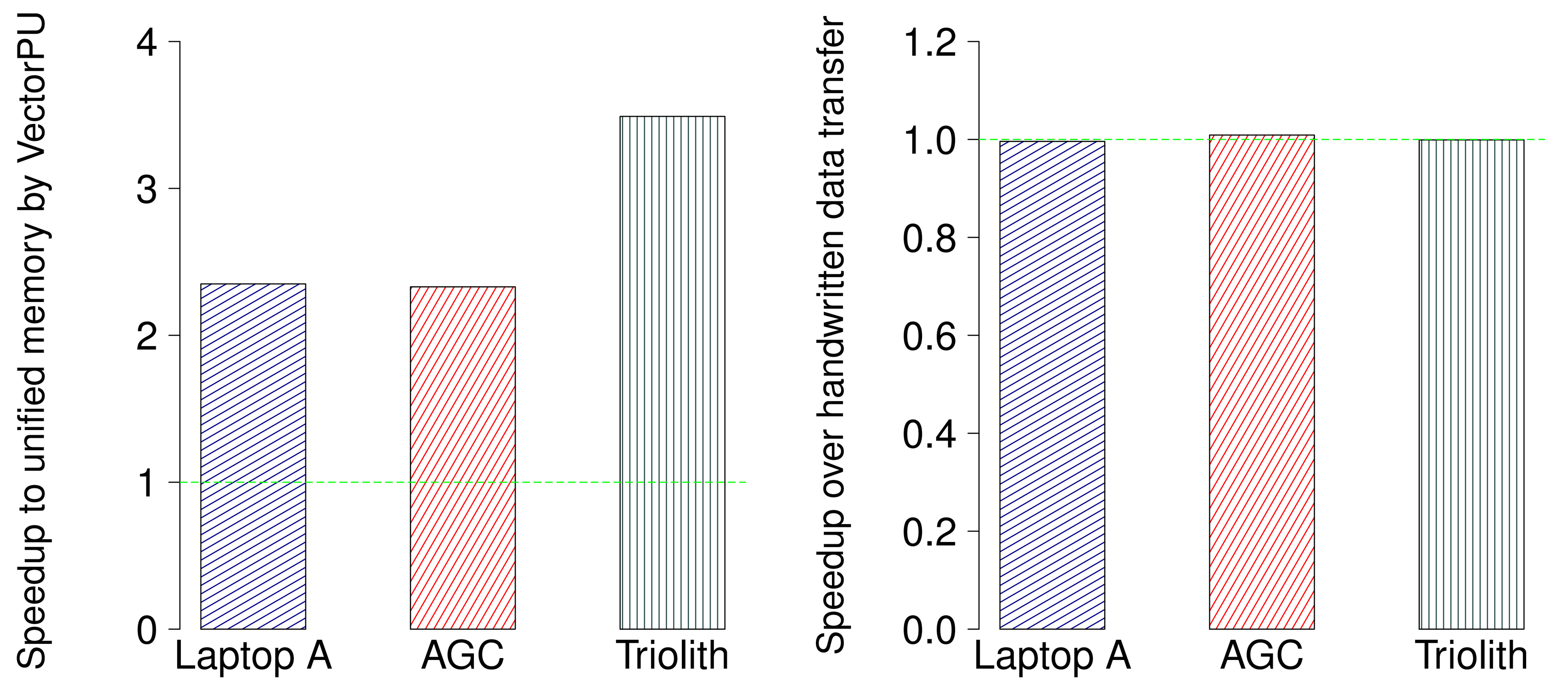
## Acknowledgments

SEVENTH FRAMEWORK PROGRAMME

EXCESS

SeRC Swedish e-Science Research Centre

## Performance Results




(a) Conjugate Gradient, compared with Nvidia's UM.

(b) FFT, compared with handwritten CUDA code.

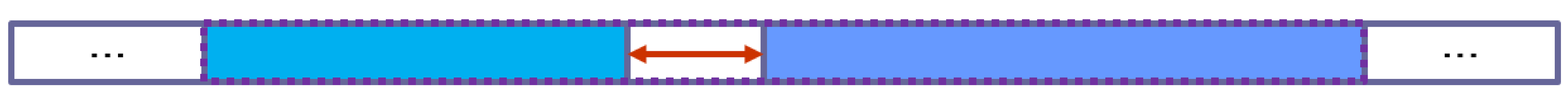
- Setup: Laptop A (laptop, Kepler GPU), AGC (workstation, Maxwell GPU), Triolith (supercomputer, Kepler GPU), CUDA 7.5
- More benchmarks compared with Nvidia's Unified Memory:
  - parallel reduction: speedup 1.40× to 8.66× on different problem sizes
  - sort: speedup 13.29× on 1M element

## Programmability Improvement

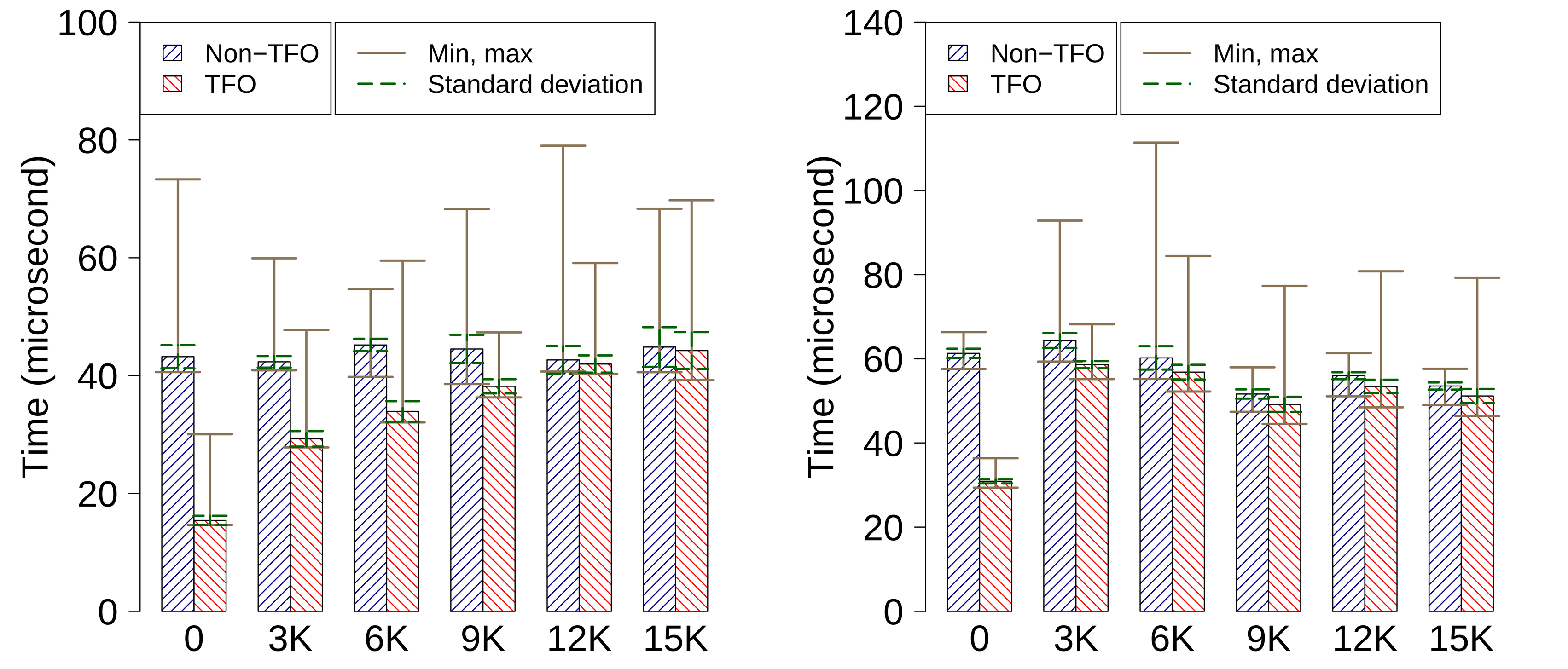
- VectorAdd from the CUDA SDK:
  - Logical LOC drops from 75 (normal CUDA program) to 24 (VectorPU)
- Parallel Reduction:
  - Logical LOC drops from 21 (Nvidia's Unified Memory) to 17 (VectorPU)

## Additional Optimizations

- **Lazy Allocation**
  - Allocations deferred until invocation points
  - Data objects to be transferred together are allocated together, so that these transfers can be fused.
  - Initially obtain speedup 2.85× by merging small data operands.




- **Transfer Fusion Optimization (TFO)**
  - Greedy TFO algorithm, proven optimal for any set of operands
  - Check at run-time the distance between operands under transfer
  - If small enough, merge the transfers by transfering redundant data between them and discard the data afterwards
  - The efficiency could be further improved in coherence management




(a) On Laptop A, speedup 1.01-2.8×

(b) On Triolith, speedup 1.05-1.98×

Figure: TFO Microbenchmark Speedups on 2 Systems. X-axis labels show gap lengths between arrays.

## Contact Information

- Open source: http://www.ida.liu.se/labs/pelab/vectorpu/
- Lu Li, Christoph Kessler