Chplx an HPX Foundation for Chapel

Shreyas Atre, Hartmut Kaiser
Louisiana State University CCT

Patrick Diehl
US Department of Energy

Christopher Taylor
Tactical Computing Labs

tactcomplabs.com

CENTER FOR COMPUTATION & TECHNOLOGY

St. George's Church
Madabah, Jordan

Overview

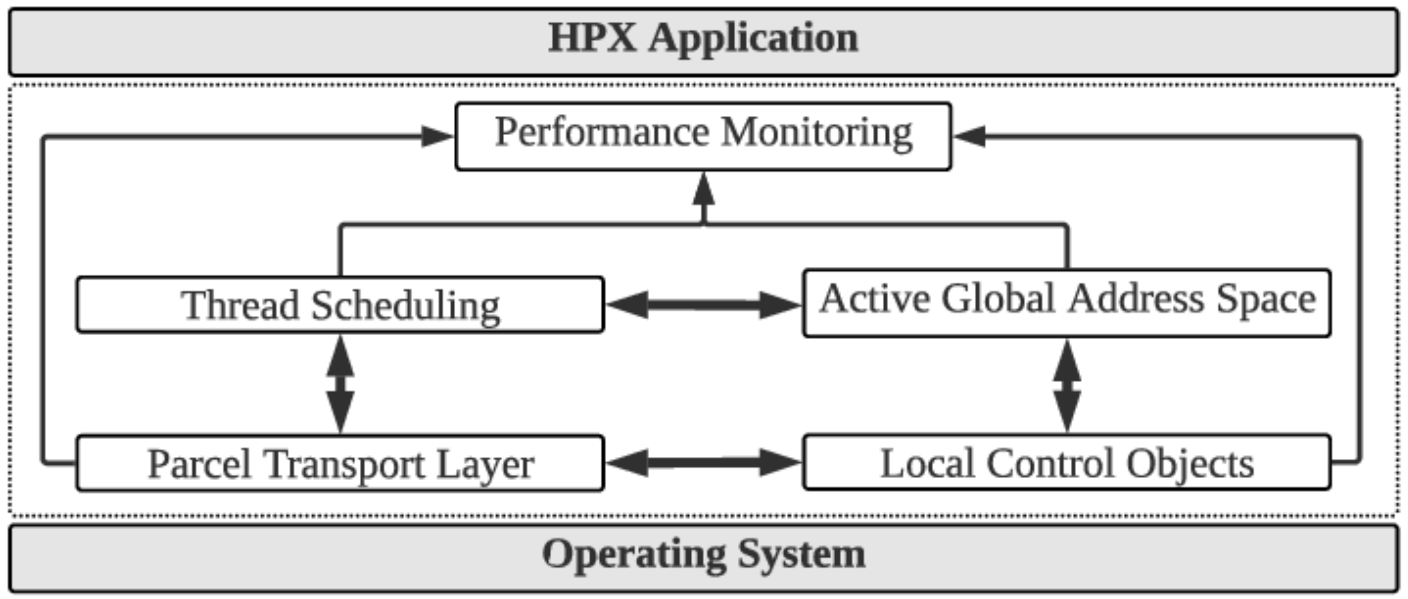
- Background HPX & Chapel
- 1D Heat Equation Study
- Chplx
- Challenges
- Benchmarks
- Analysis

Runtime Systems?

- Runtime systems are systems level software that organize hardware and operating system level services
  - HPC application software attempts to minimize operating system dependencies/interactions
- Runtime systems are designed around different abstraction models
  - Charm++ uses the Actor abstraction model
  - UPC++ has it’s own unique abstraction model
  - HPX uses the ISO C++ data parallelism and concurrency model
  - Chapel’s runtime system abstraction model is coupled to programming language features

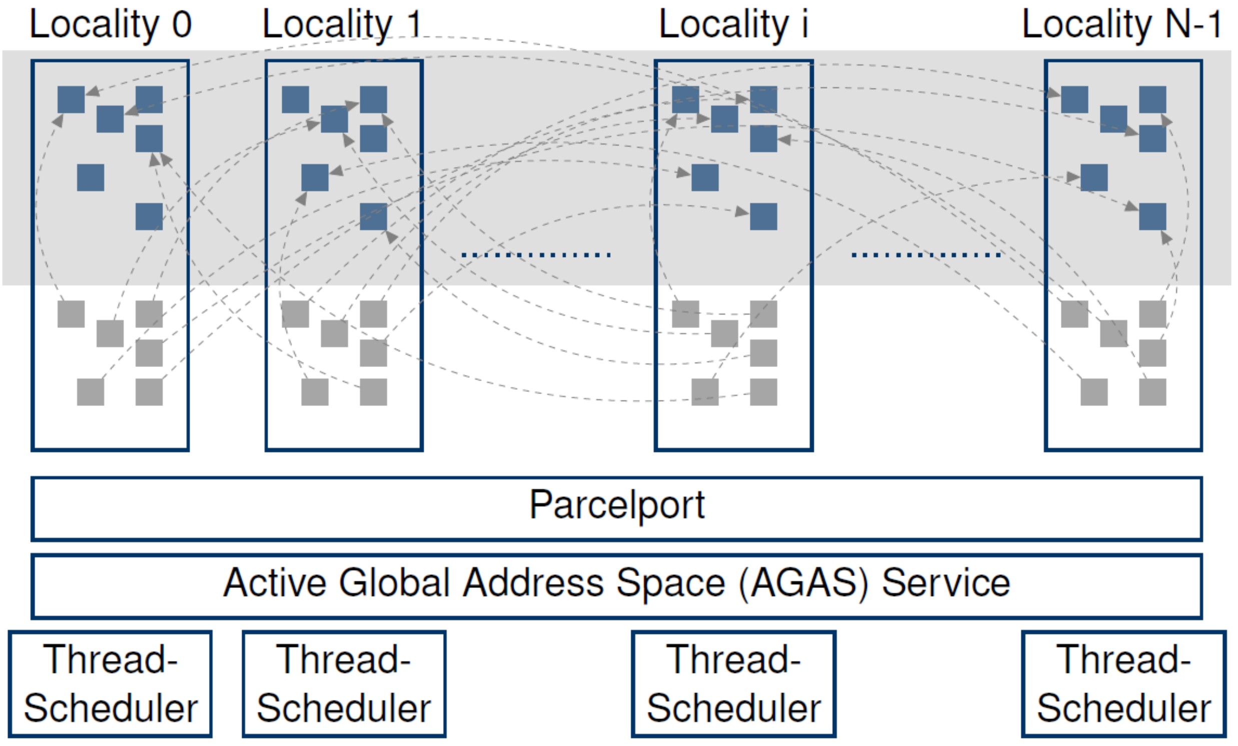
HPX

- HPX is an Asynchronous Many Task Runtime System
- All features housed under the ISO C++ standard for data parallelism and concurrency
- User's develop ISO C++ complaint code, using an API that mirrors the ISO C++ STL
- All functionality is provided "for free"
- Emphasis on *futures* and *futurization* - C++ DAGs chained together w/futures



HPX

- Objects (partitioned data types) can be instantiated across different localities - AGAS (Active Global Address Space)
- Asynchronously deploy remote functions and methods yielding a form of 1 and 2 sided communication (active messages)
- Supports SPMD / non-SPMD
- Parcelport: OpenMPI, LCI, libfabric, sockets, OpenSHMEM*, GASNet*
- Implements support for OpenMP



Chapel

- Programming Language supporting PGAS - Partitioned Global Address Space
- Users program the runtime system by-way of Chapel’s *language features*
  - Parallelism through explicitly defined control structures in the Chapel’s grammar; SPMD style loops, data parallel thread level loops, etc
- Distributed data types instantiated using ‘Domains’
  - Users can predefine data layouts and tiling strategies making array-based distributed programming more approachable

Chapel

- Productivity oriented language
  - Feels like a scripting language but compiles out to statically optimized executables
- Provides support for distributed arrays (variation on Coarrays/ZPL) and distributed data structures
- Places substantial emphasis on Domain (index set) loop structures
  - Users *could* cross a physical boundary by not using domains and adversely impact performance
  - ISO C++ Ranges are a conceptually similar abstraction

HPX & Chapel

- HPX & Chapel have more in common than they are different
  - Venn diagram shows significant functionality overlap
- Chapel provides a level of accessibility and convenience at the language layer of the application software stack which would benefit HPX application development

1D Heat Equation Study

1D Heat Equation Study

CENTER FOR COMPUTATION & TECHNOLOGY

Tactical Computing Labs

"Benchmarking the Parallel 1D Heat Equation Solver in Chapel, Charm++, C++, HPX, Go, Julia, Python, Rust, Swift, and Java"

Diehl, Brandt, Morris, Gupta, Kaiser

Euro-Par 2023 : Parallel Processing Workshops

Pre-print: https://arxiv.org/abs/2307.01117

1D Heat Equation Study

CENTER FOR COMPUTATION & TECHNOLOGY

Tactical Computing Labs

- 1 Dimensional Heat Equation

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}, \quad 0 \leq x < L, t > 0,$$

- alpha term is material diffusivity
- 2nd Order differencing
- Euler's method

$$u(t + \delta t, x_i) = u(t, x_i) + \delta t \cdot \alpha \frac{u(t, x_{i-1}) - 2 \cdot u(t, x_i) + u(t, x_{i+1})}{2h}$$

Chapel	Julia
C++	Python
Charm++	Rust
HPX	Swift
Go	Java

1D Heat Equation Study

- Software Complexity Measured
- **Co**nstructive **Co**st **Mo**del (COCOMO)
  - `scc` - *Sloc Cloc and Code*
  - Estimated Schedule Effort (ESE)
    - Time to implement in a month

1D Heat Equation Study

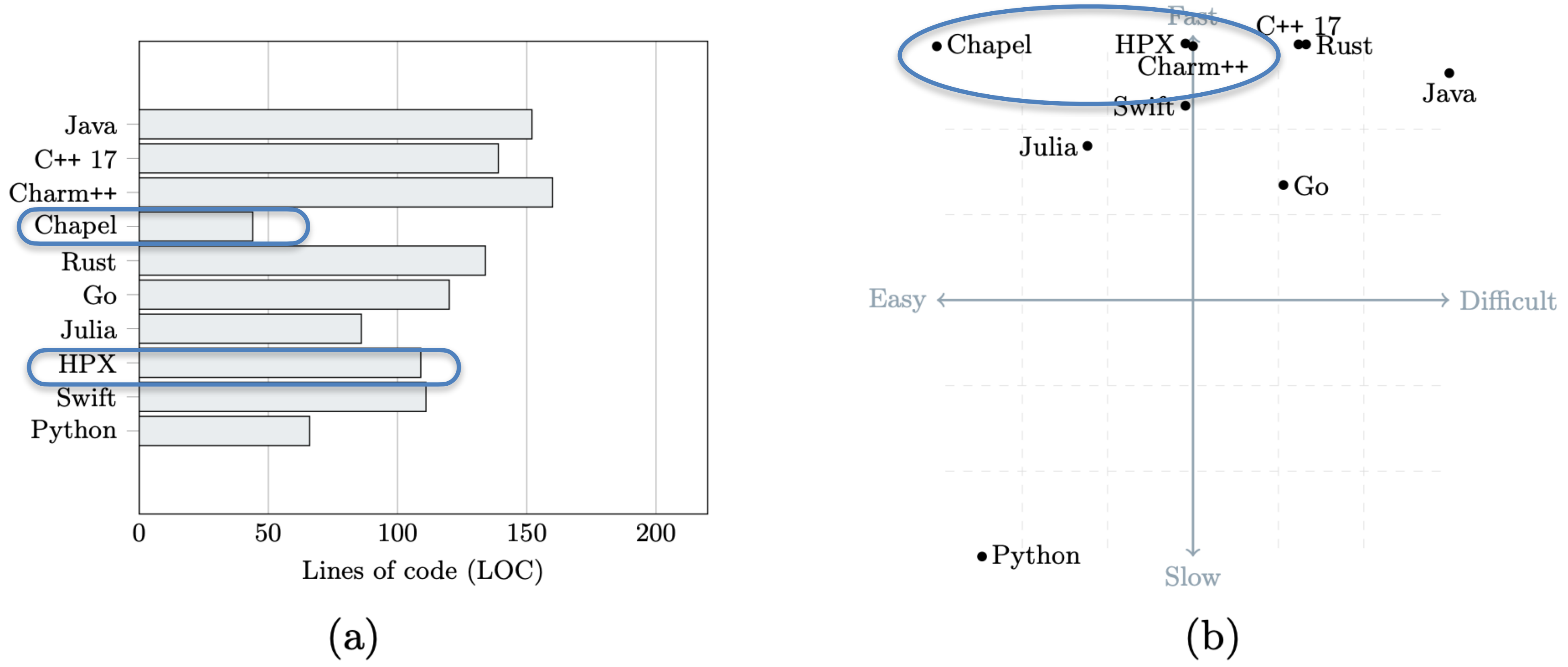


Fig. 1. Software engineering metrics: (a) Lines of codes for all implementations. The numbers were determined with the Linux tool *cloc* and (b) Two-dimensional classification using the computational time and the COCOMO model.

Chplx

Chplx

- Follow on study
  - Can the complexity gap b/n HPX & Chapel be closed?
  - Source to source compilation! *Chapel to C++*
  - Heat Equation, STREAM, & GUPS
  - What could be done in ~6 months of part time effort?

Chplx

- Use HPE/Cray-Chapel compiler's existing lexing and parsing infrastructure
- Chapel's language features implemented as an ISO C++20 library using HPX
- Support enough Chapel to generate C++ for each identified benchmark
- Focus on single-locality solution

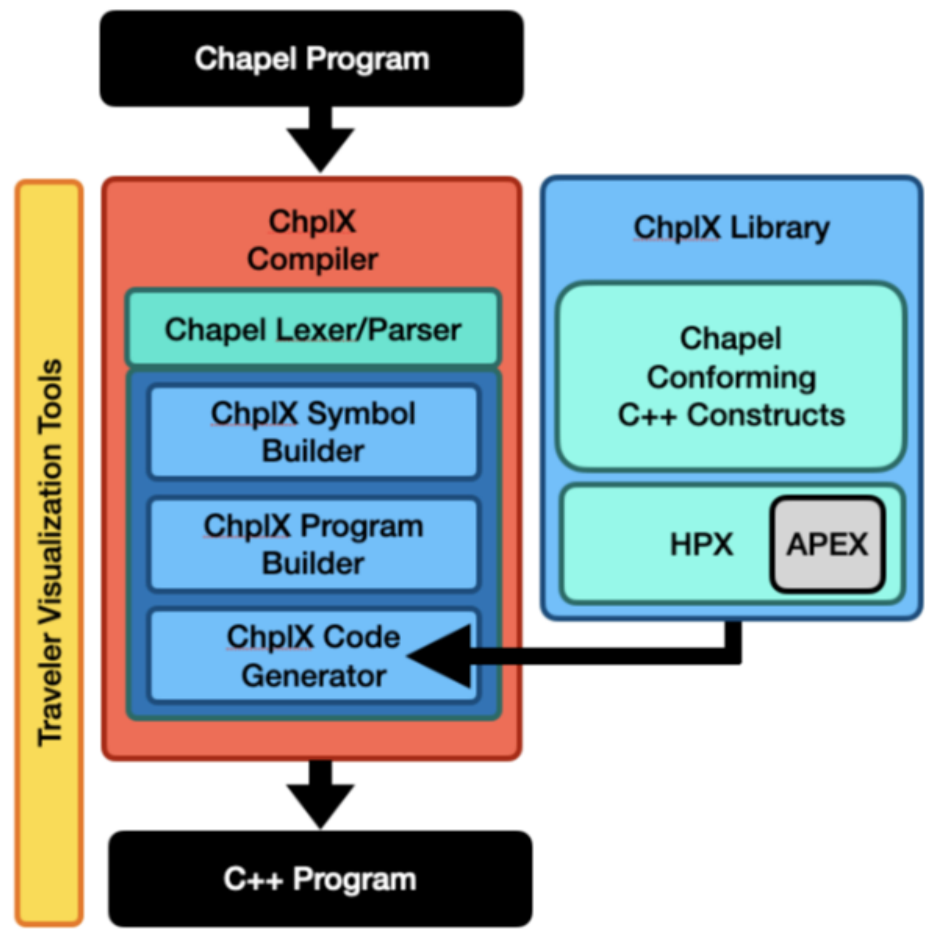


Figure 1: The ChplX Toolchain

Chplx

- The Chplx compiler has *3 passes*
  - Symbol Table/Scope Creation
  - Chapel AST to Chplx Program Tree
  - C++ Code generation w/CMake support

Chplx

- No type checking!
  - Allow the C++ compiler to manage that w/pragmas
  - Users can select a C++ compiler
    - This study uses Clang for consistency

Challenges

Challenges

LSU CENTER FOR COMPUTATION & TECHNOLOGY

TCL Tactical Computing Labs

- HPE/Cray-Chapel compiler's parse tree (uAST)
  - Naively used uAST
  - uAST's structure injects Scope AST nodes into the program structure
  - Complicates Syntax analysis
  - Scope handling in this scenario is challenging

Table 1: Chapel language Conditional Expressions in the AST, '{}' represents a scope, 'cond' represents the conditional

Chapel Conditional	Chapel AST

`if () {}`
`else if () {}`
`else {}`
	cond() {
	{
	cond() {}
	{}
	}
	}

`if () {}`
`else if () {}`
`else if () {}`
	cond() {
	{
	cond() {
	{
	cond() {}
	}
	}
	}
	}

Challenges

- Chapel support for print statements and timers?
  - Created a *new intrinsic function* for Chapel to inline C++ `inlinecxx()`
  - Provides pass-through so users can embed C++ code into a Chapel application
  - Uses ISO C++20 support for `std::fmt`

Listing 1: ChplX 'inlinecxx' input and compiled C++ output

```
// `inlinecxx` function signature
proc inlinecxx(string, n?...)

// Chapel code that uses `inlinecxx`
var i = 0;
inlinecxx("std::cout << i << std::endl");
inlinecxx("std::cout << {} << std::endl", i);

// ChplX generated C++ output from lines 5-6
int i = 0;
std::cout << i << std::endl;
std::cout << i << std::endl;
```

Benchmarks

Benchmarks

- Intel(R) Xeon(R) CPU E5-2680 w/2.5 GHz
  - 48 cores, 128 GiB DDR4
  - Ubuntu focal
  - Clang+LLVM 15**, HPX v1.9.1
  - Chapel 2.0

CHPL_RT_NUM_THREADS_PER_LOCAL

--hpx:threads

** Chapel parser/lexer requirement

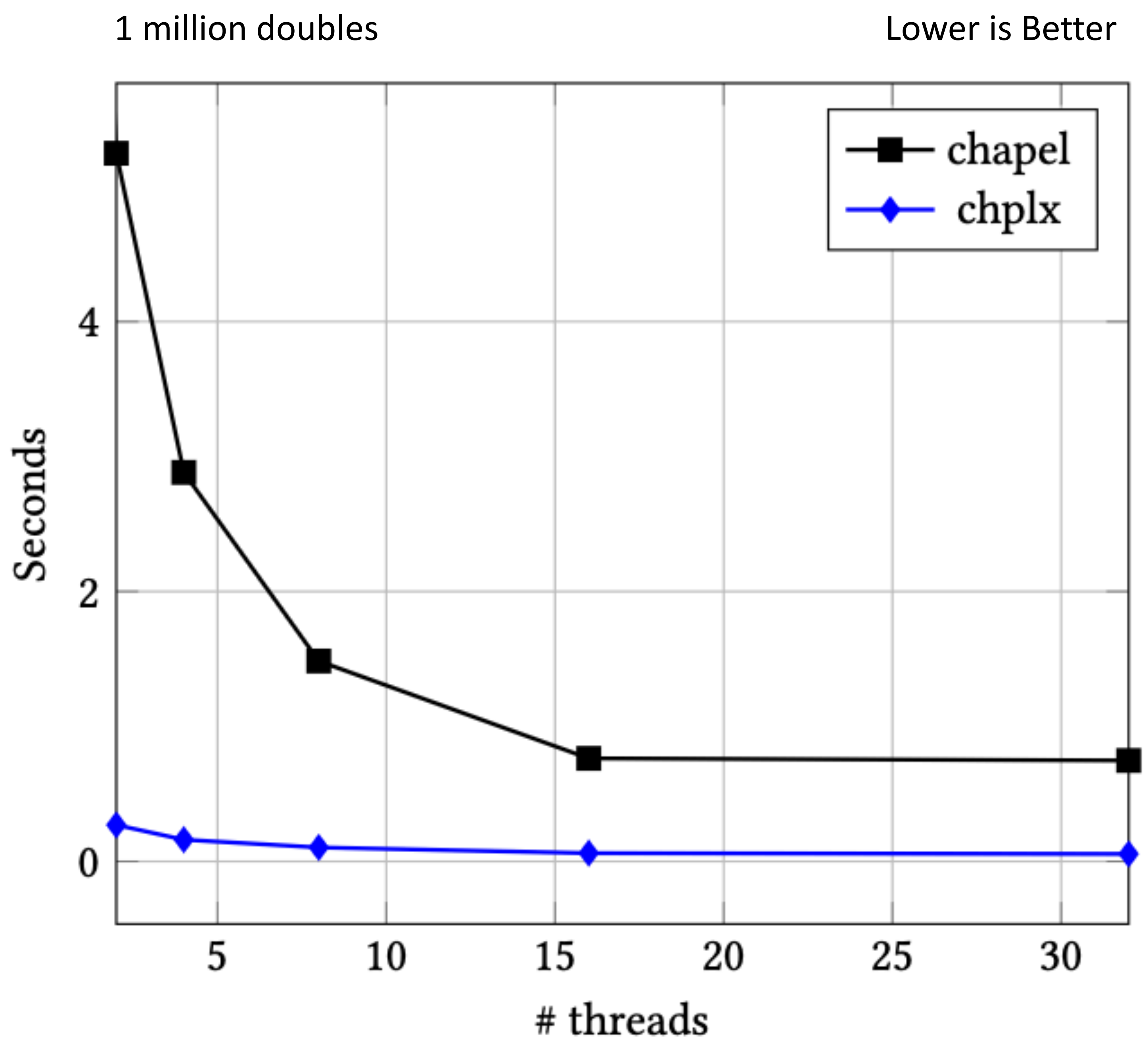
Benchmarks

CENTER FOR COMPUTATION & TECHNOLOGY

Tactical Computing Labs

- Chplx
  - Generates C++ and the project CMake files
  - `-std=c++20` enabled for ISO C++ coroutine support
    - ISO C++20 coroutines provide `co_await` & `yield` functionality
    - C++ `co_await` & `yield` are equivalent to python generators

Benchmarks - 1D Heat Eqn

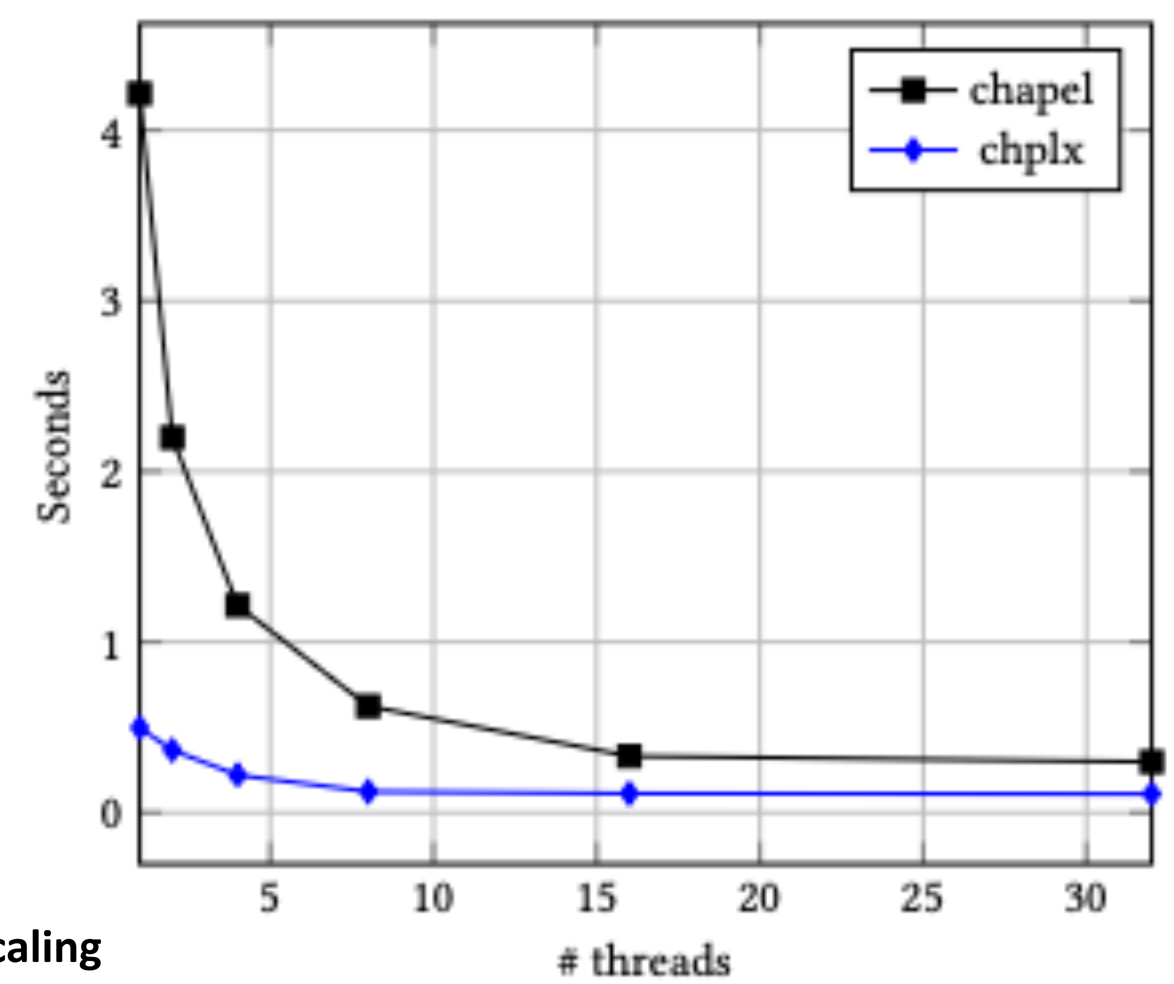


Heat Equation: Strong Scaling, Average Time

Benchmarks - STREAM Copy

LSU CENTER FOR COMPUTATION & TECHNOLOGY

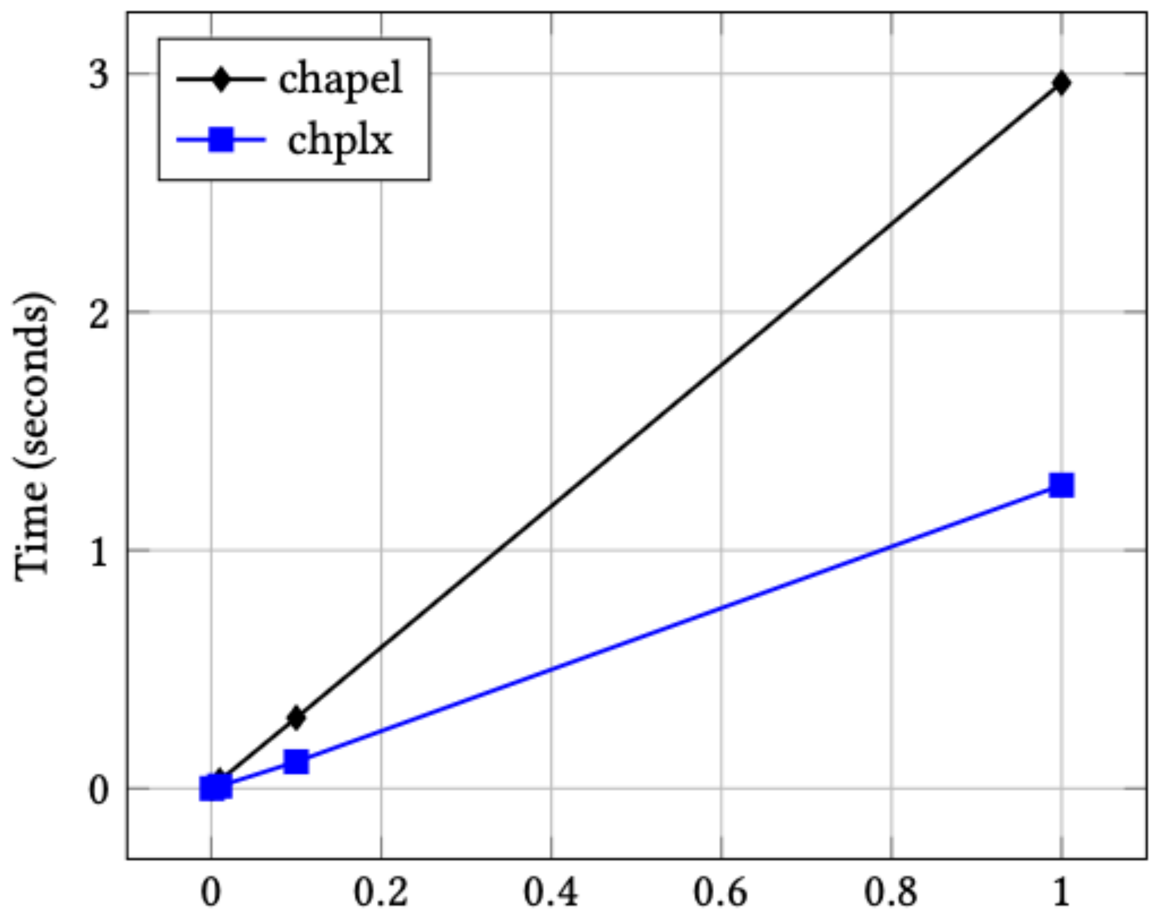
Tactical Computing Labs



Strong Scaling

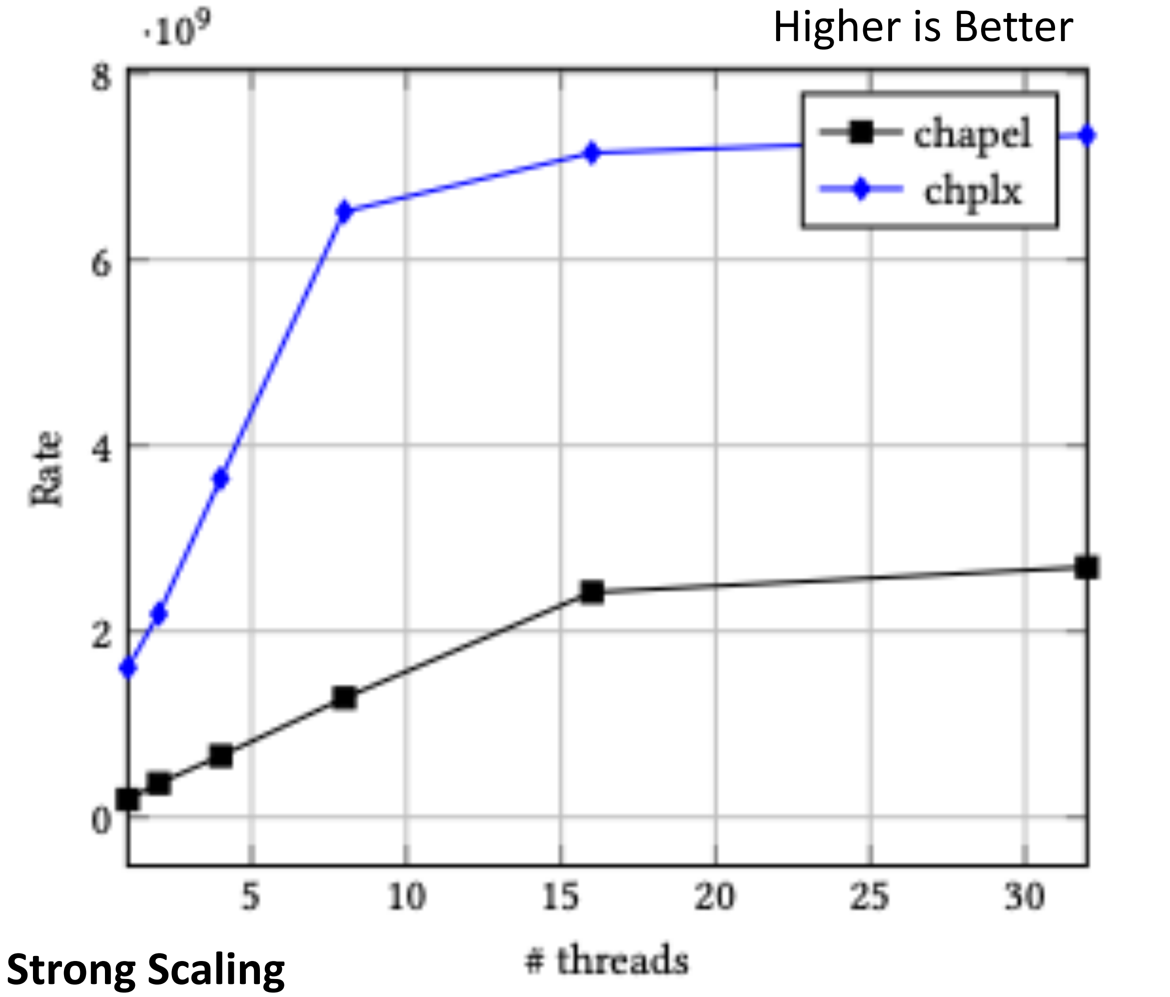
STREAM Copy for $10^8 real(64)$

1 million doubles



Megabytes

STREAM Copy data scaling over 32 threads



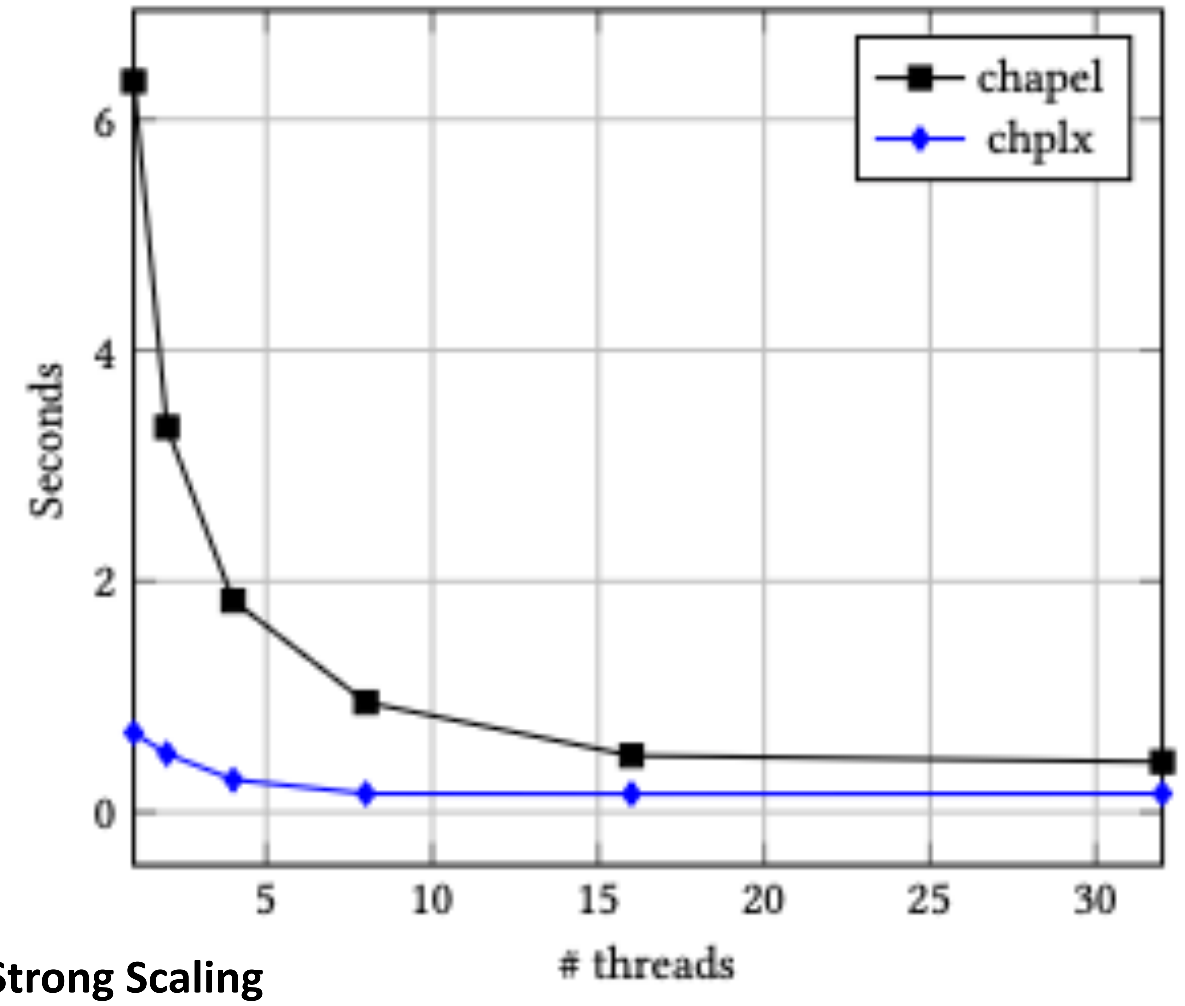
Strong Scaling

STREAM Copy for $10^8 real(64)$

bytes / time

Benchmarks - STREAM Add

Lower is Better

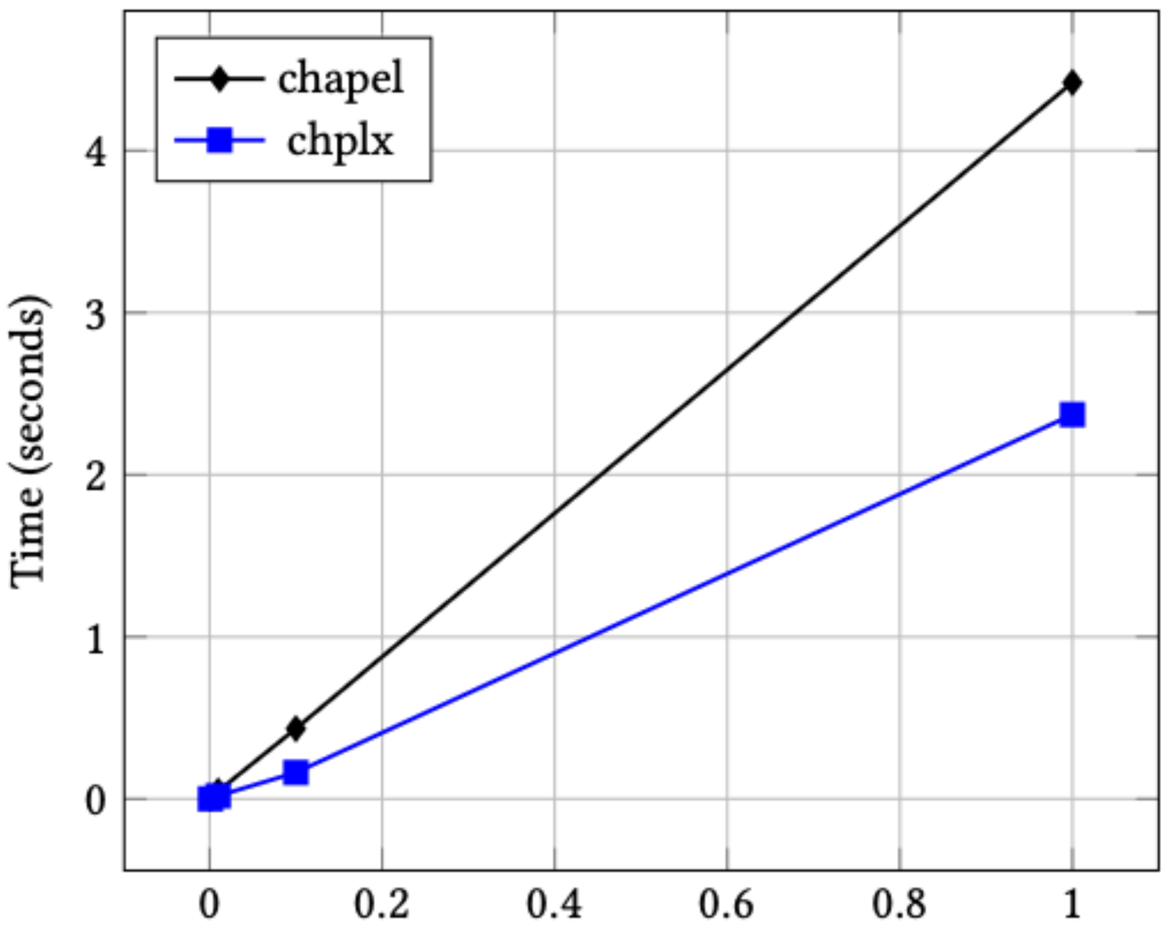


Strong Scaling

STREAM Add for 10^8 *real*(64)

1 million doubles

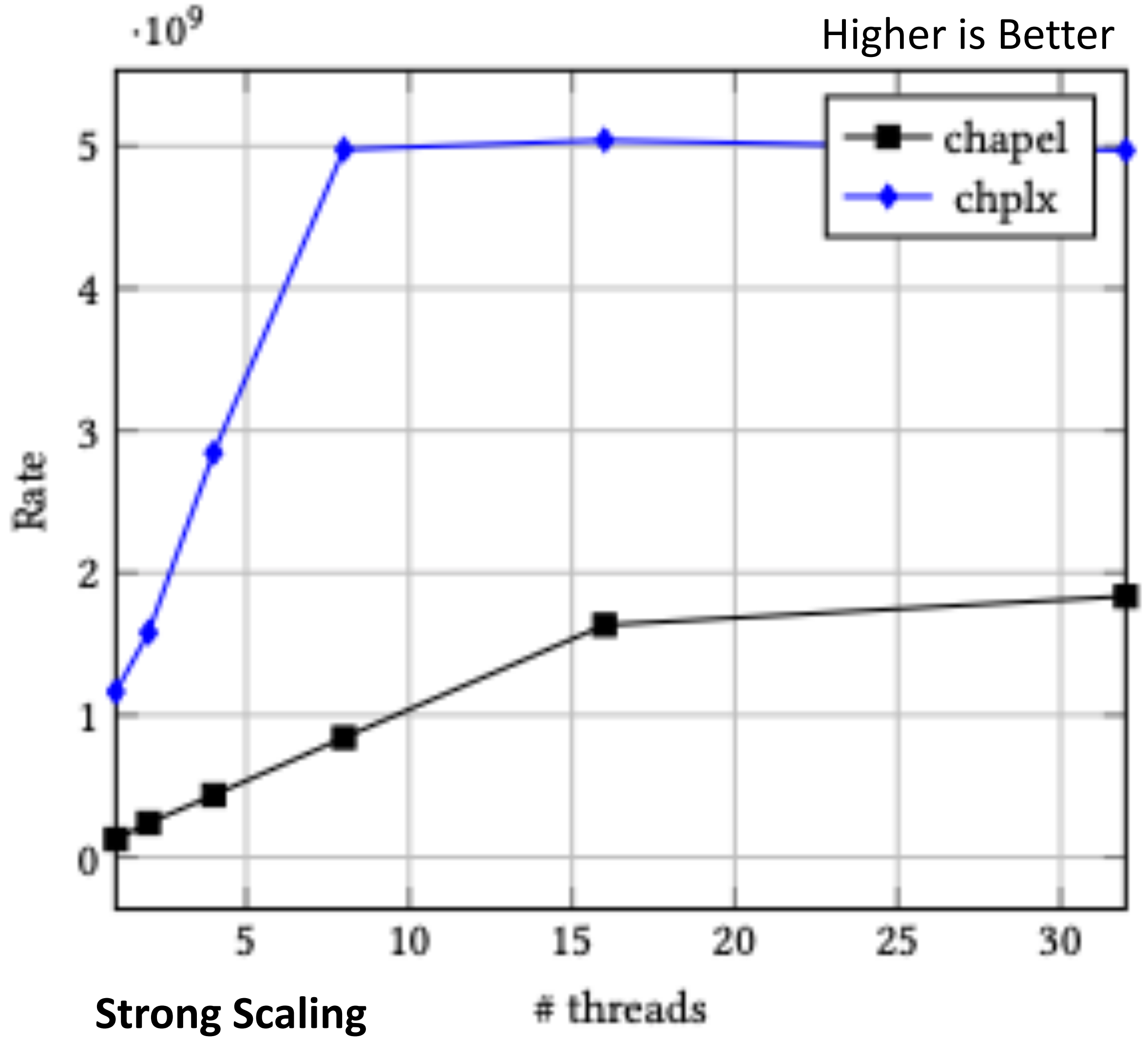
Lower is Better



Megabytes

STREAM Add data scaling over 32 threads

Higher is Better



Strong Scaling

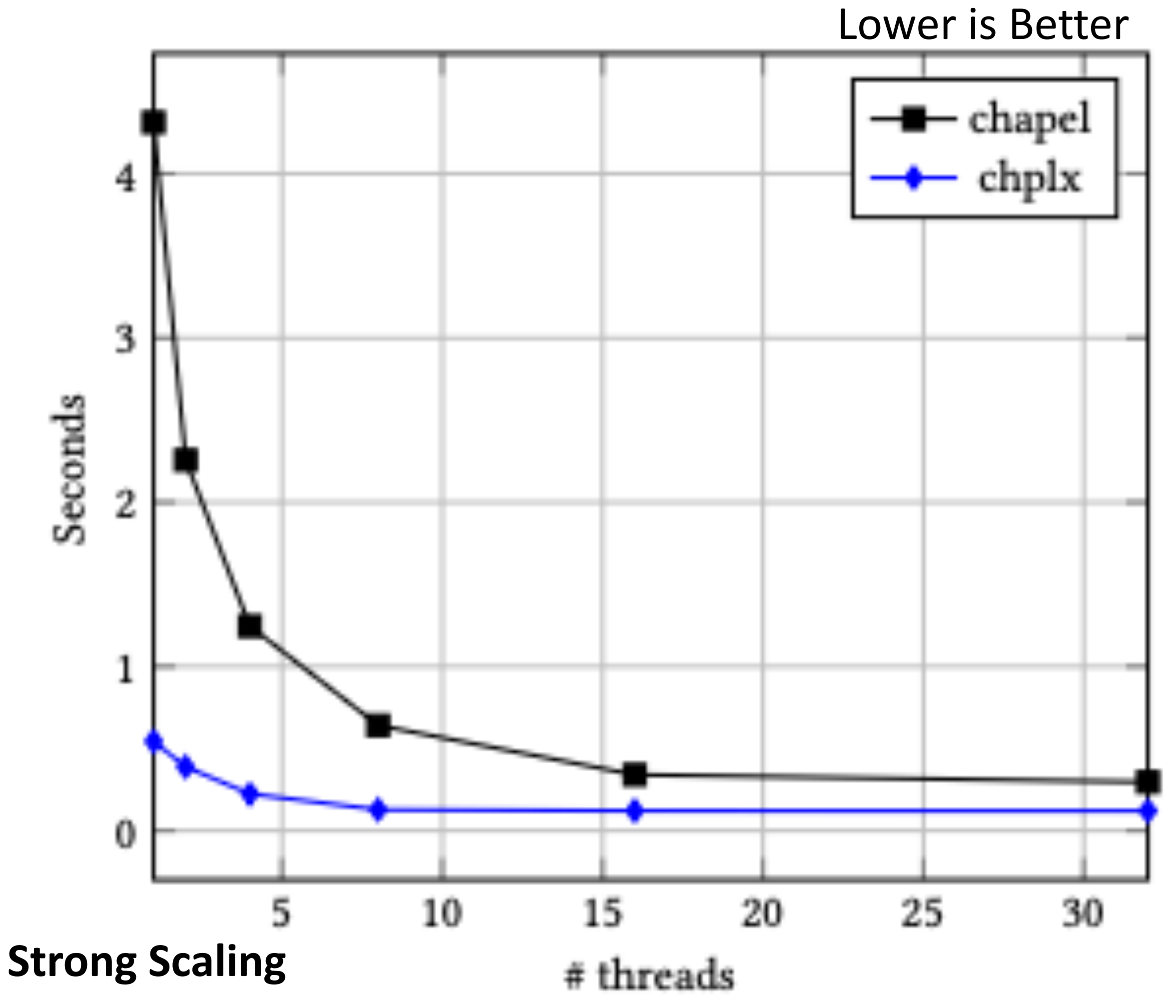
STREAM Add for 10^8 *real*(64)

bytes / time

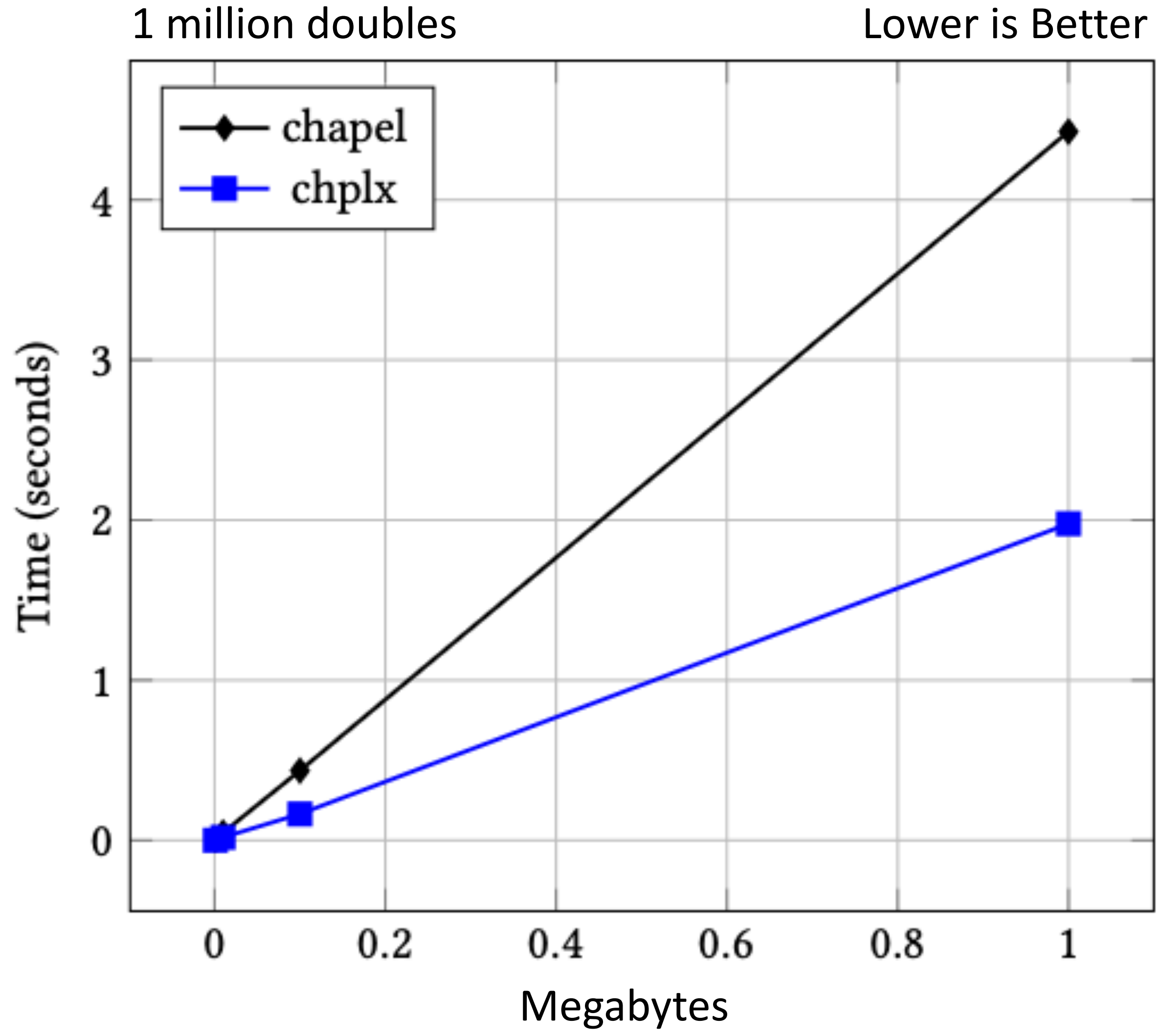
Benchmarks - STREAM Scale

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

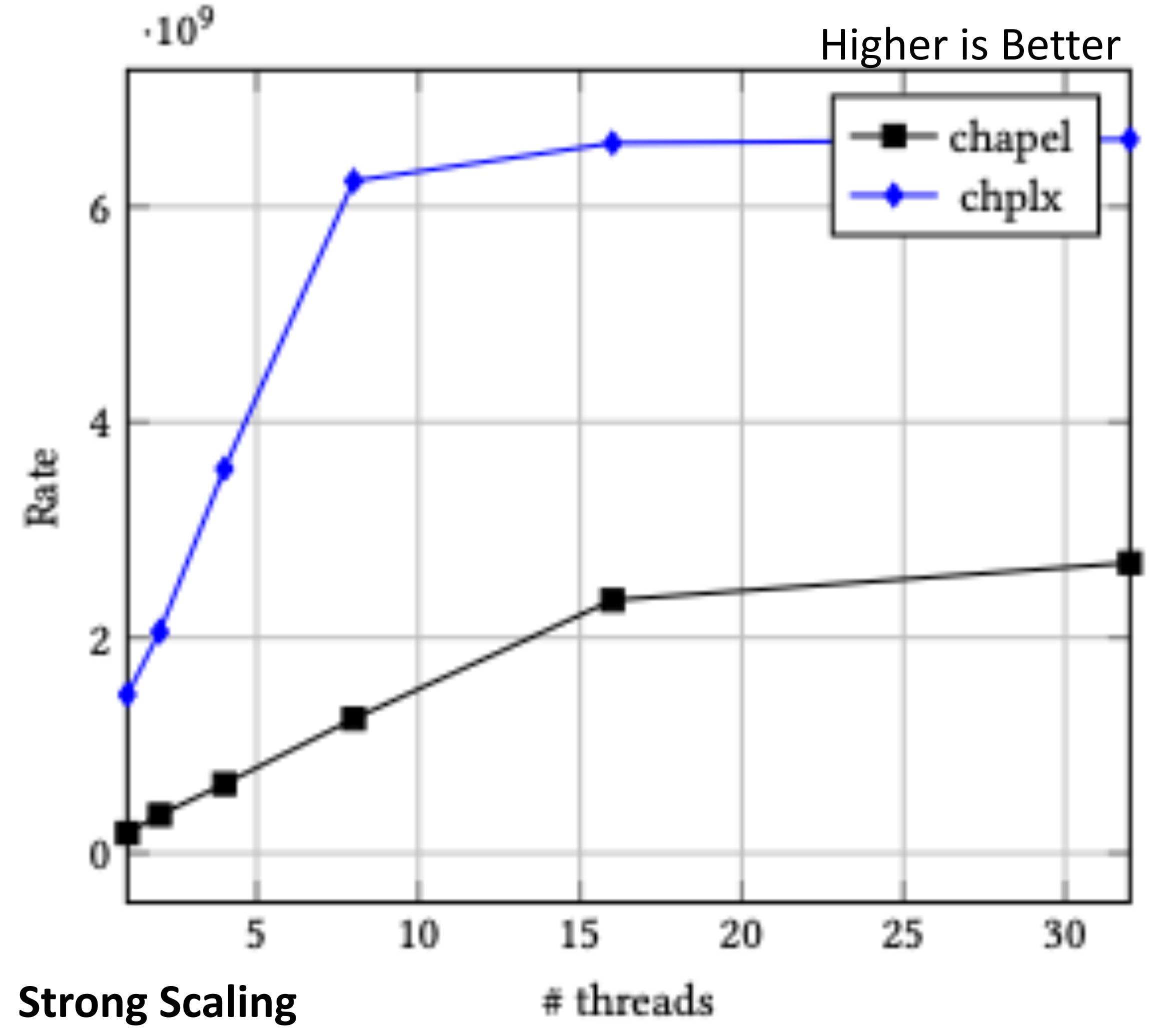
Tactical Computing Labs



STREAM Scale for $10^8 real(64)$



STREAM Scale data scaling over 32 threads



STREAM Scale for $10^8 real(64)$

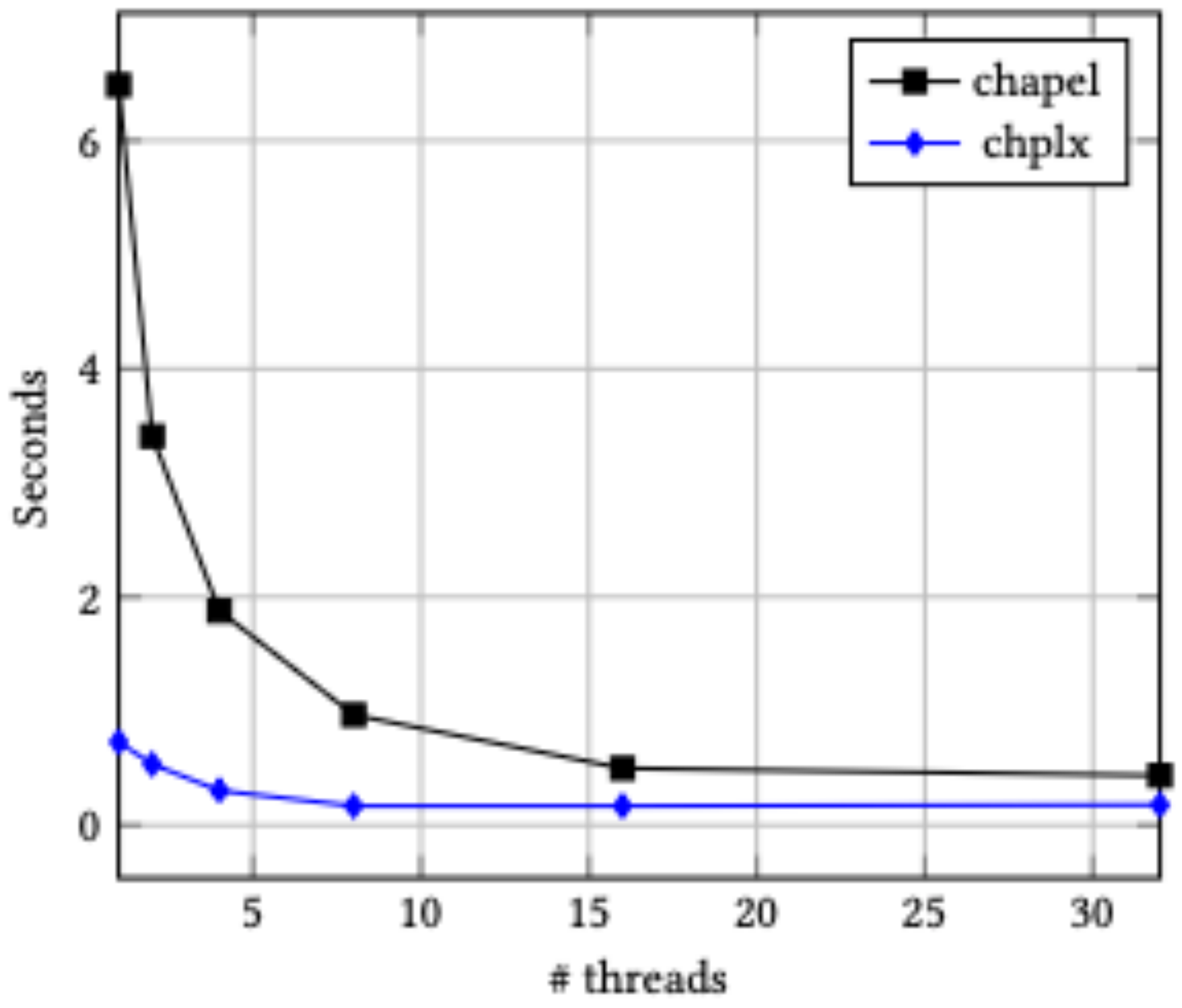
bytes / time

Benchmarks - STREAM Triad

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

TCL Tactical Computing Labs

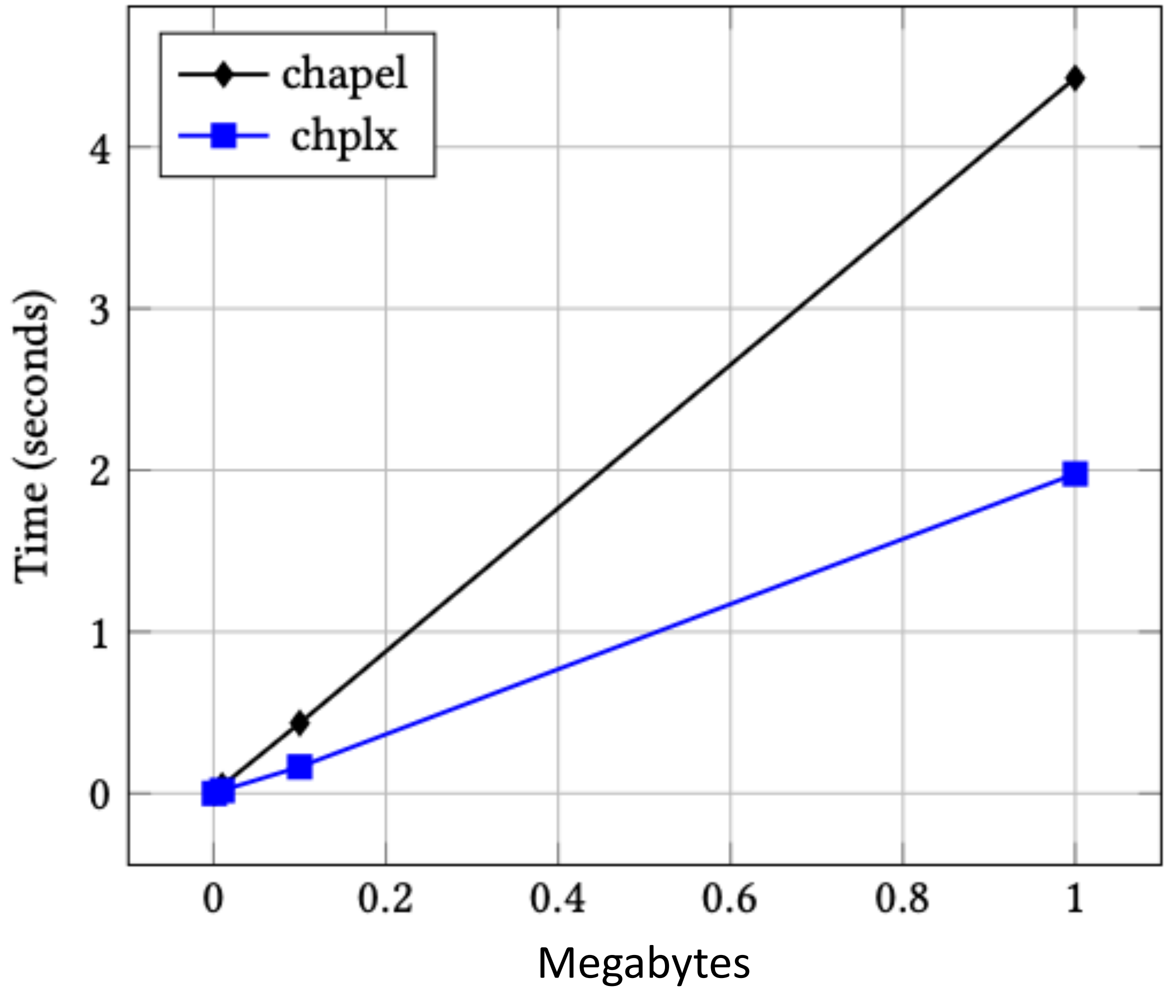
Lower is Better



STREAM Triad for $10^8 real(64)$

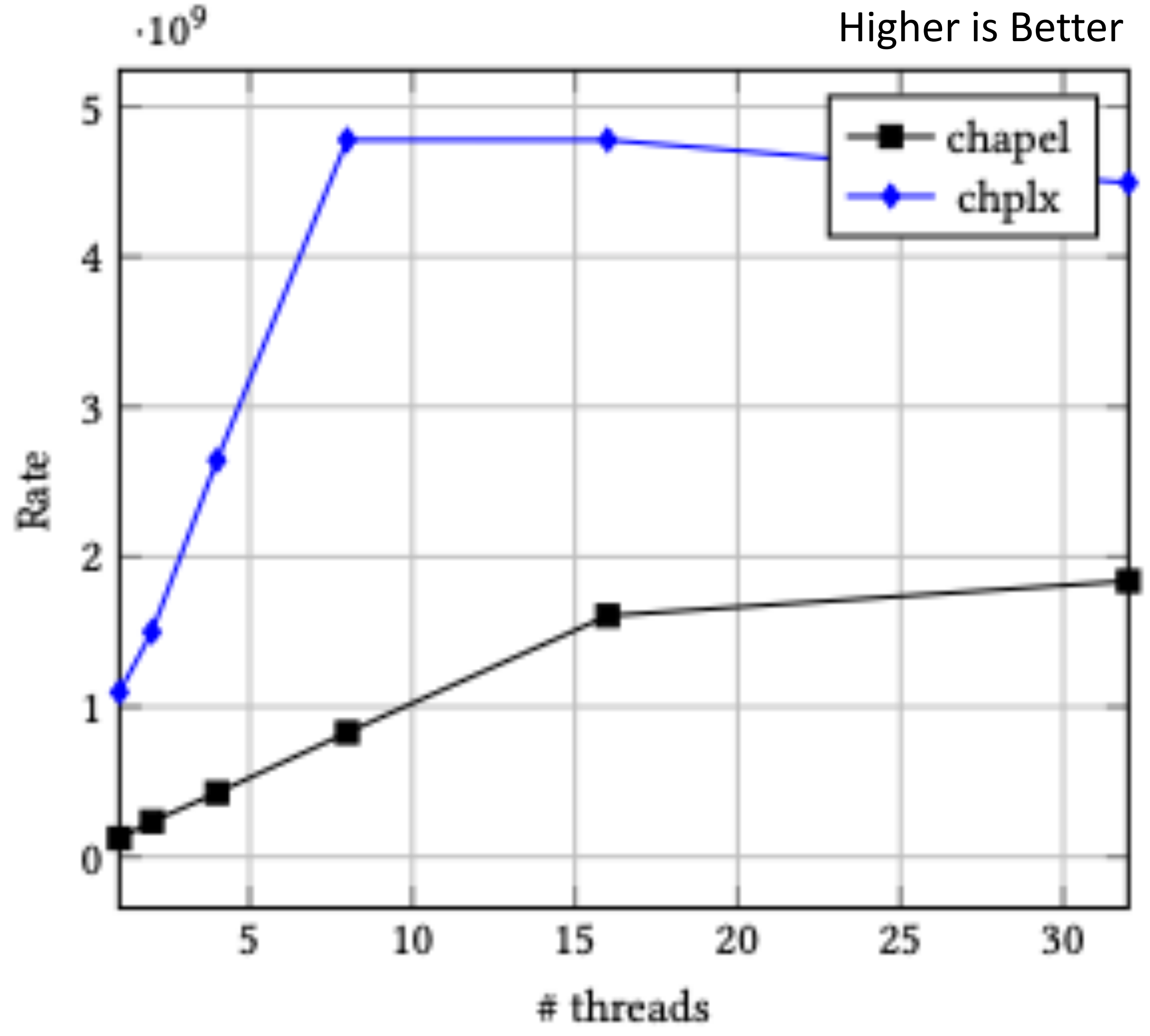
1 million doubles

Lower is Better



STREAM Triad data scaling over 32 threads

Higher is Better

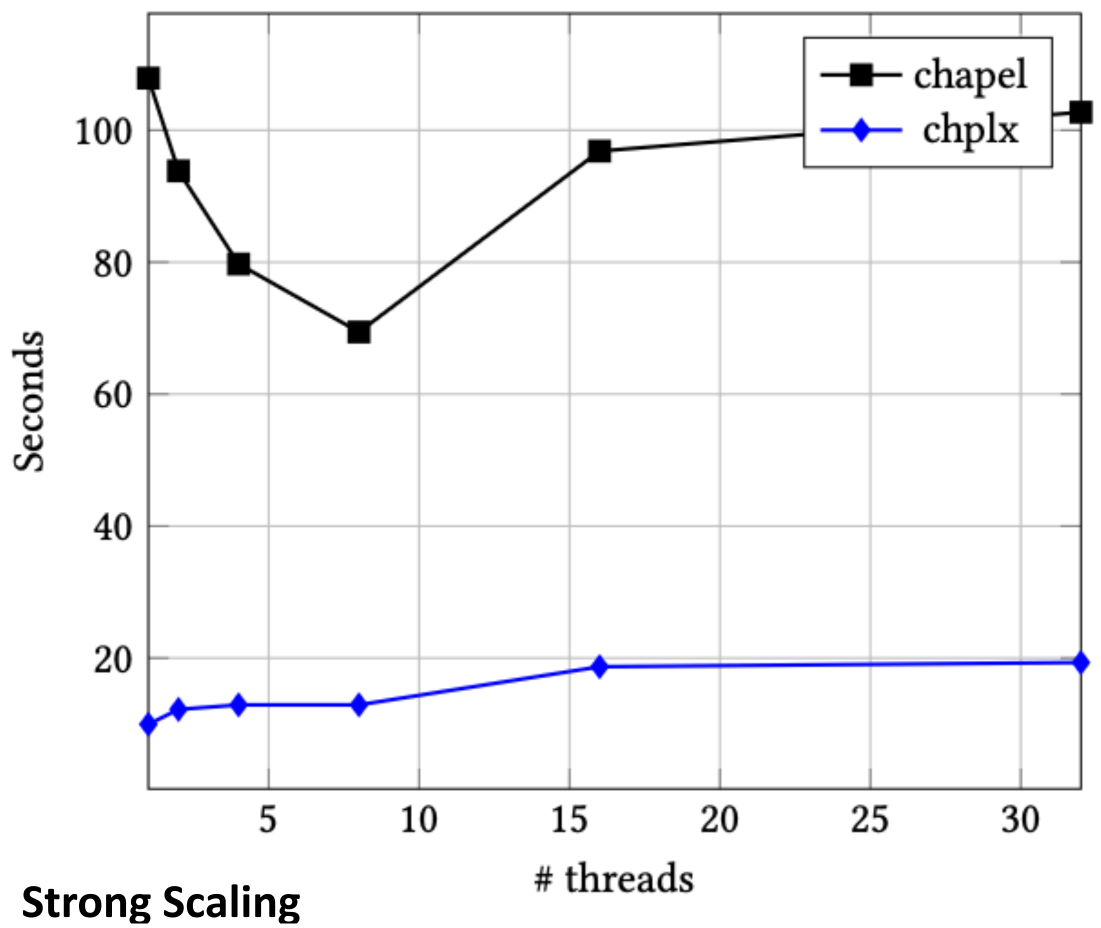


STREAM Triad for $10^8 real(64)$

bytes / time

Benchmarks - GUPS

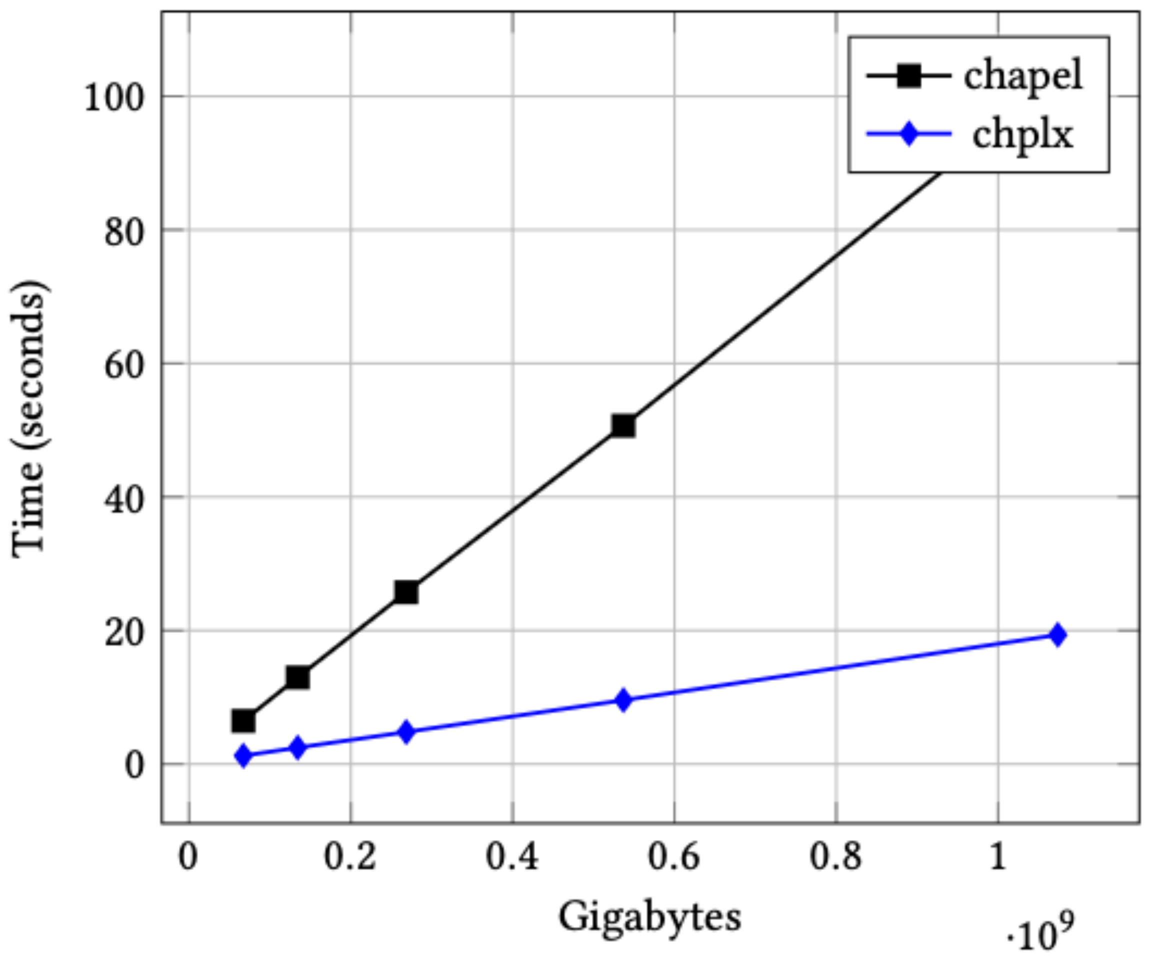
Lower is Better



Strong Scaling

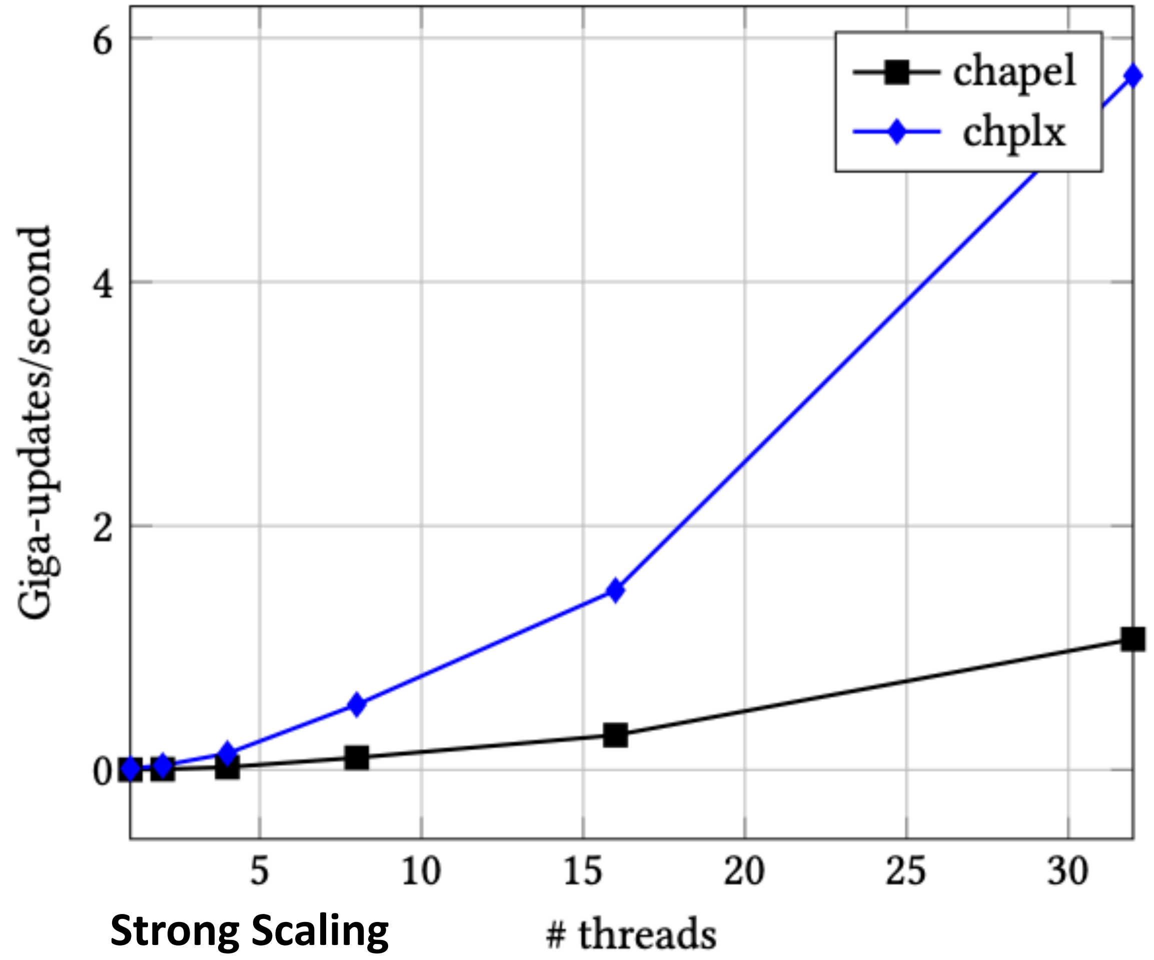
GUPS for 1GB of 17GB Average

Lower is Better



GUPS data scaling over 32 threads

Higher is Better



Strong Scaling

GUPS for 1GB of 17GB, Average

num_updates = n_threads * bytes
gups = num_updates / seconds / 1e9 (gig)

Benchmarks - COCOMO

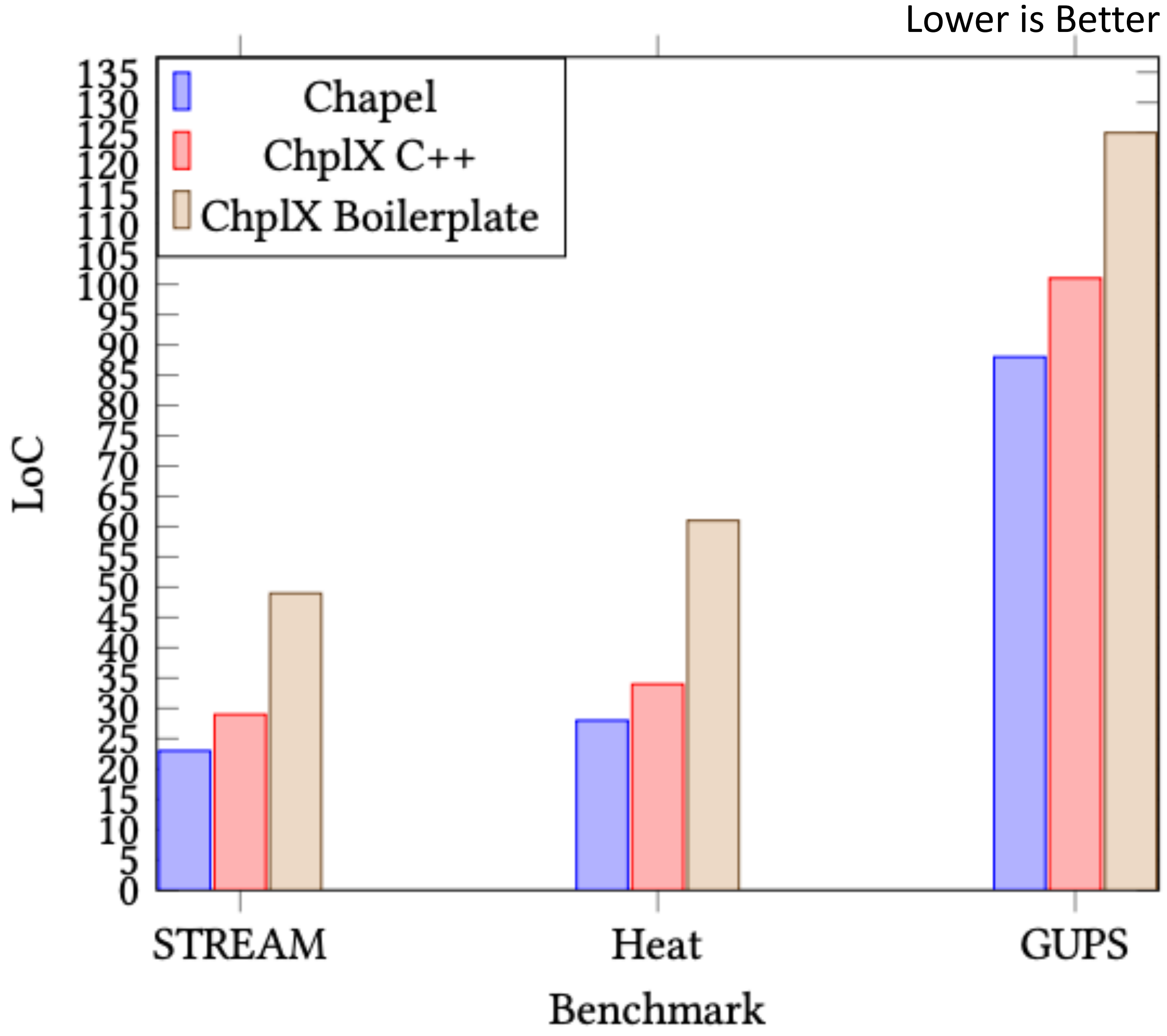


Figure 8: Lines of Code Measurements

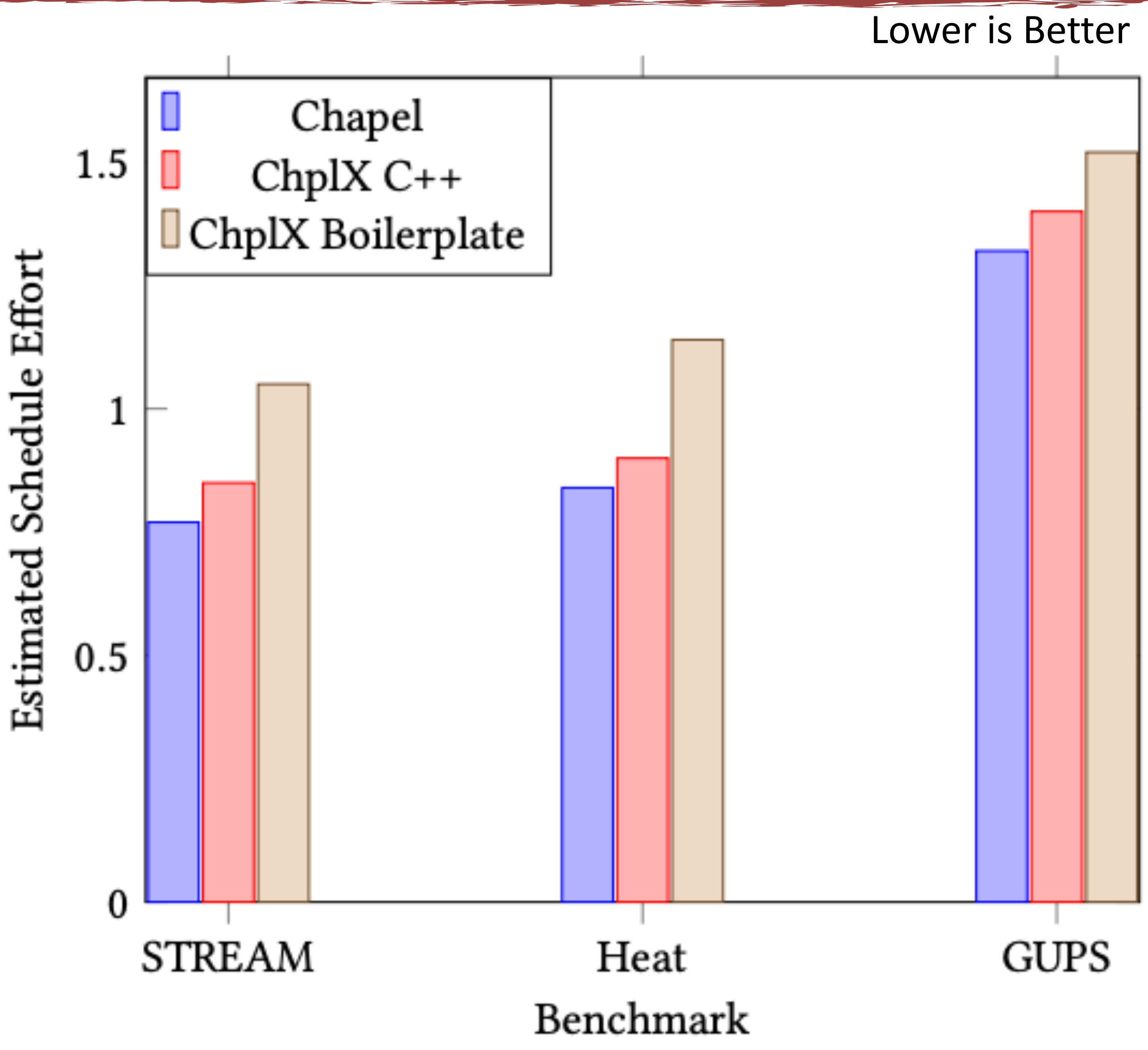


Figure 9: COCOMO: Estimated Schedule Effort

Analysis

Analysis

- Performance delta?
  - Generated code
  - Data copies?
  - Runtime system
    - Chapel w/qthreads
    - Chplx w/HPX

Analysis

LSU CENTER FOR COMPUTATION & TECHNOLOGY

TCL Tactical Computing Labs

- COCOMO differential
  - Did not achieve 1-to-1 mapping…got *closer*
  - +/- 4 Lines of Code w/o Boilerplate
  - Not bad considering the time investment

Analysis

- Given the code complexity gap closed and the performance offered by Chplx
- Effort demonstrates a viable avenue of continued research!
  - Resolve uAST issue
  - Develop support for multi-node distributed computing

Clip Art License Compliance Statement

- The clip art in this presentation was created by rawpixel.com

Thanks!

LSU

CENTER FOR COMPUTATION
& TECHNOLOGY

TCL

HPX

STE||AR GROUP