

Enhanced Data and Task Abstractions for Extreme-scale Runtime Systems

PhD Thesis by Nick Vrvilo
Department of Computer Science
Rice University



Committee: Vivek Sarkar, Corky Cartwright and Lin Zhong

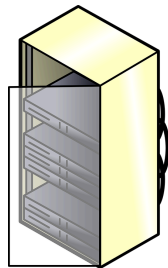
June 30, 2017

Extreme-scale Computing

- 100-way parallelism on a chip
- 1000-way parallelism on a node
- Similar energy footprint to current systems



exascale
in a data center



petascale
in your lab



terascale
on your desk



gigascale
wearables

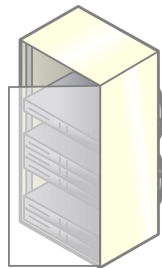
Challenges at Extreme-scale

- Need 100x more parallelism in software
- Highly constrained memory & bandwidth
- Frequent failures



exascale

in a data center



petascale

in your lab



terascale

on your desk



gigascale

wearables

Extreme-scale Runtimes

- New runtimes for extreme-scale:
 - HPX
 - Open Community Runtime (OCR)
 - Realm
- Existing runtimes adapting to extreme-scale:
 - Berkeley UPC
 - Charm++

*My contributions are marked in
GREEN*

Runtime Feature Comparison

	<i>Pointer Safety</i>	<i>Blocking</i>	<i>High-level languages</i>
Charm++	Programmer's responsibility	ucontext	Not yet supported
HPX	Programmer's responsibility	ucontext or boost::context	Not yet supported
OCR	Static and dynamic checks	Many options presented	CnC
Realm	Programmer's responsibility	ucontext	Legion, Regent
UPC++	No data migration	SPMD blocking	Not yet supported

OCR: Runtime Goals

- *Portability* for apps across hardware
- *Resilience* as a fundamental design feature
- *Performance* through hints and introspection



Note: *programmability* is not a goal of OCR.

OCR: Programming Model

- *Tasks* for computation
- *Datablocks* for all non-temporary data
- *Events* for dependence management



OCR: Ecosystem Vision

C, C++, Fortran

R-Stream, ROSE, LLVM

CnC, Chapel, ...

Legion, HCLib,
Habanero-UPC++, ...

Hero
Programmer

Smart
Compiler

Higher-level
language

Higher-level
library

Open Community Runtime Framework
(Building Blocks)

External Runtime Components

MPI, GASNet,
OpenSHMEM,
UCX, ...

Extreme-scale Platforms



OCR: Intended Users

- Concurrency experts / “hero programmers”
- Library implementers
- Compiler back-end engineers



Note: *mere mortals* are not the target users.

Thesis Statement

We assert that runtime challenges tied to extreme-scale computing can be solved with marginal overhead, while also limiting the burden placed on the application programmer.

Our Proposed Solutions

1. Position-independent object encoding for migratable datablocks
2. Practical support for blocking constructs in lightweight tasking runtimes
3. CnC-OCR: a productivity layer for OCR

#1 Position-independent object encoding for datablocks

C, C++, Fortran

R-Stream, ROSE, LLVM

CnC, Chapel, ...

Legion, HCLib,
Habanero-UPC++, ...

**Hero
Programmer**

Smart
Compiler

Higher-level
language

Higher-level
library

Open Community Runtime Framework
(Building Blocks)

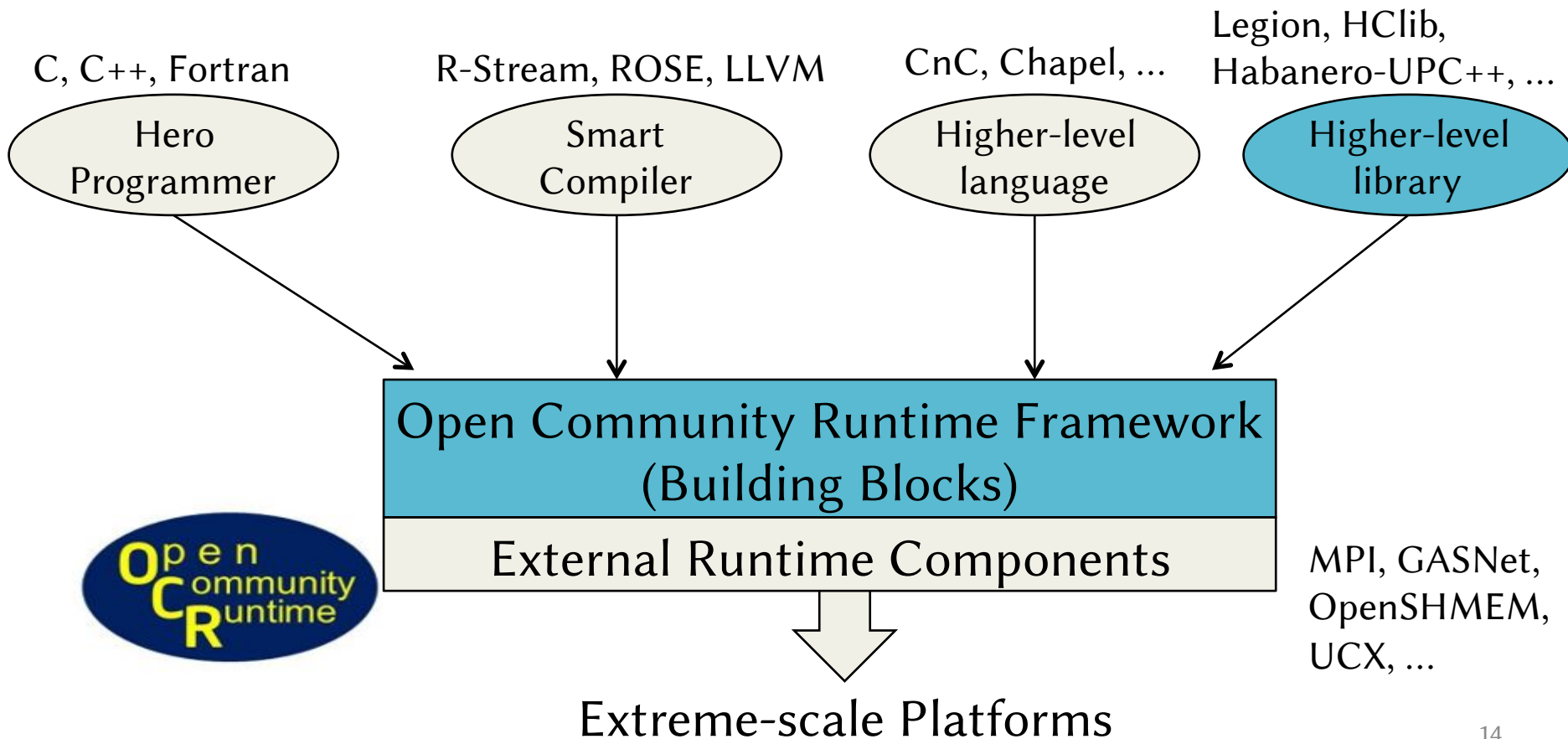
External Runtime Components

MPI, GASNet,
OpenSHMEM,
UCX, ...

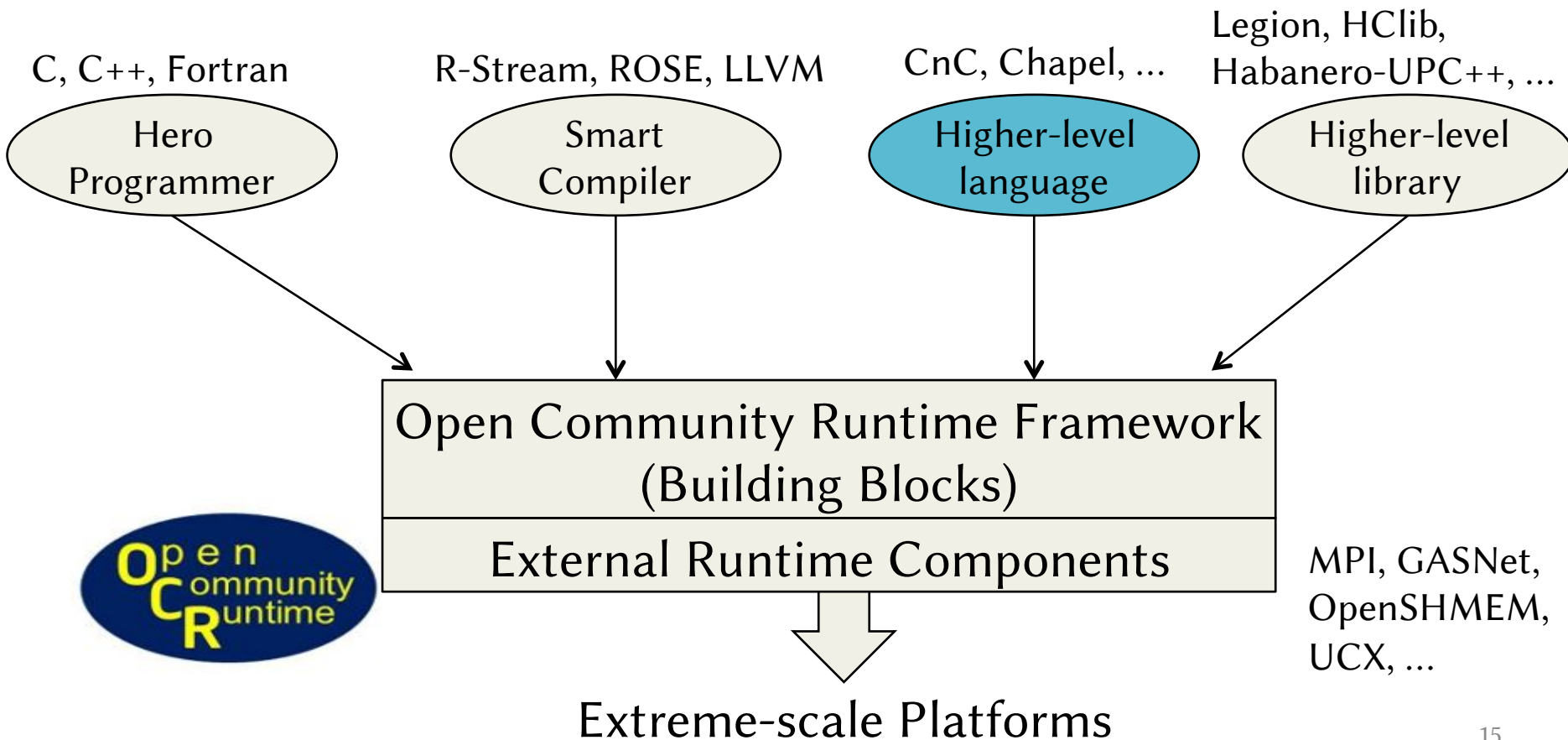
Extreme-scale Platforms



#2 Practical support for blocking constructs



#3 CnC-OCR: a productivity layer for OCR



Outline

1. Position-independent object encoding for migratable datablocks
2. Practical support for blocking constructs in lightweight tasking runtimes
3. CnC-OCR: a productivity layer for OCR
4. Conclusions and future directions

C++ Gaining Ground in HPC

- AllScale (EU Horizon 2020)
- Kokkos (Sandia)
- Legion (Los Alamos & Stanford)
- RAJA (Lawrence Livermore)
- UPC++ (Berkeley)

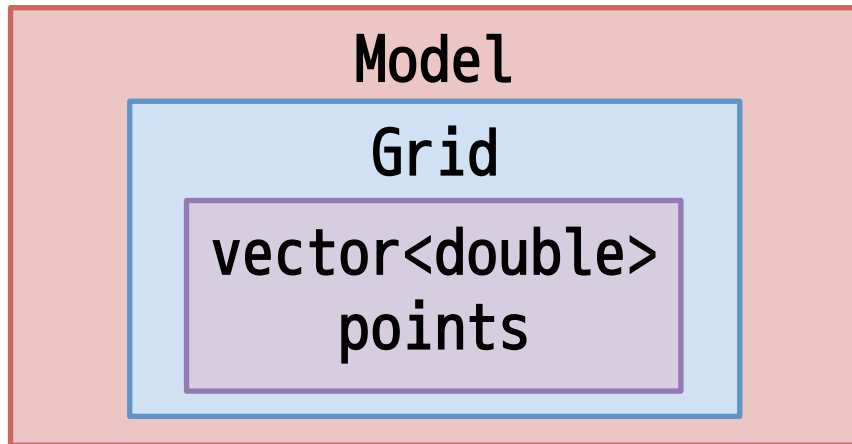
... but OCR only defines a C language API

Problem #1: Description

- Requirements of OCR data model:
 - OCR can move datablocks (whenever not in use)
 - OCR treats datablocks as opaque (memcpy contents)
 - Moving invalidates pointers into a datablock
 - All persistent data must be stored in datablocks
- Consequence on application code:
 - Native pointers cannot be persisted across tasks!

Motivating C++ App: Tempest

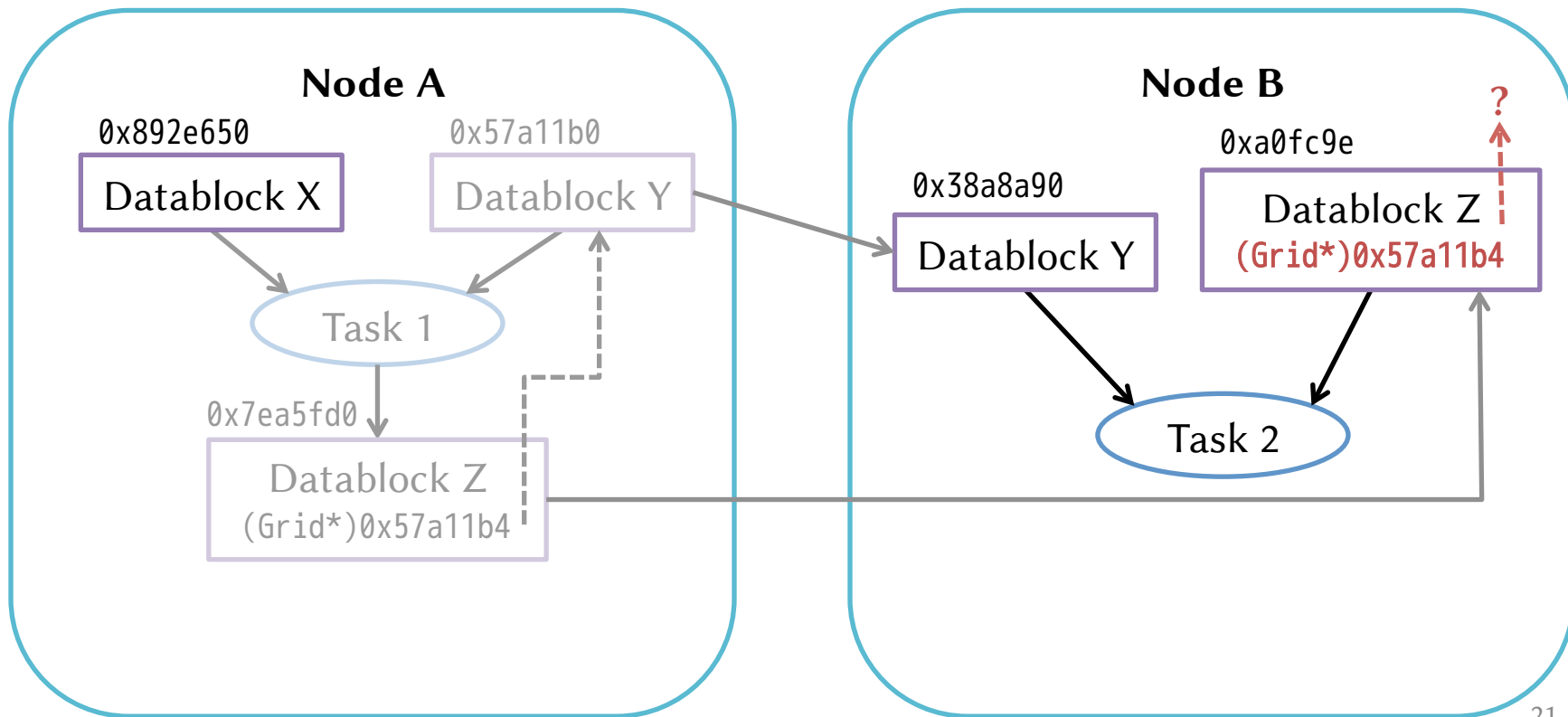
- Climate system modeling framework
- Lead developer: Paul Ullrich (UC Davis)
- Uses idiomatic C++:
 - Aggregate objects
 - STL vectors
- Port to OCR started by Gabriele Jost



OCR + Tempest: Problems

- Tempest uses C++...
⇒ `extern "C"`, `static_cast` from `void*`, etc.
- Tempest expects point-to-point messages
⇒ Rewrite blocking code (manual CPS-transform)
- Tempest model uses aggregate C++ objects
⇒ ??? (segfaults in distributed runs)

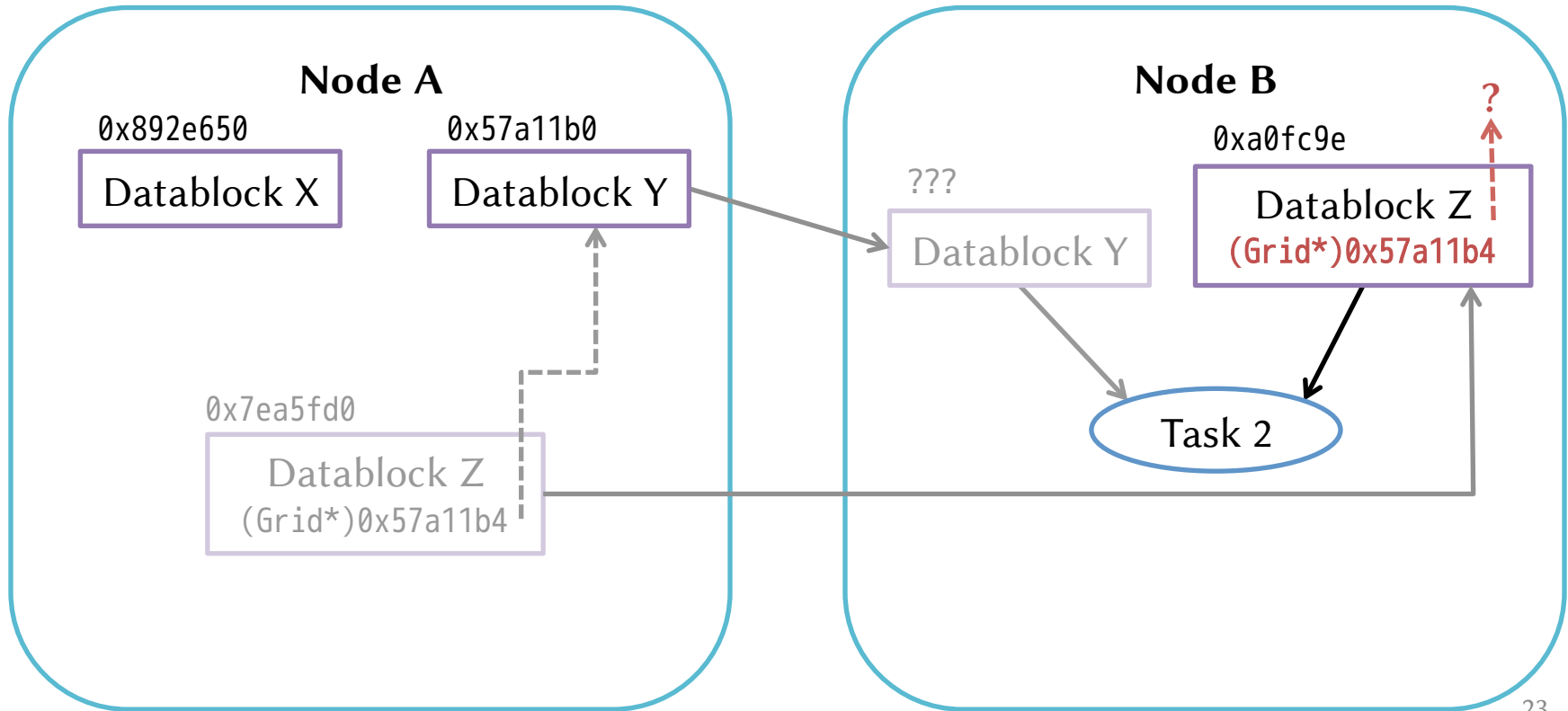
Problem #1: Example



No Serialization Support in OCR

- OCR does not support object serialization
 - No pre/post-migration hooks for datablocks
 - Pure C API also makes serialization difficult
- Assume we added pre/post migration hooks
 - Can correctly update intra-datablock pointers
 - **Still can't handle inter-datablock pointers**

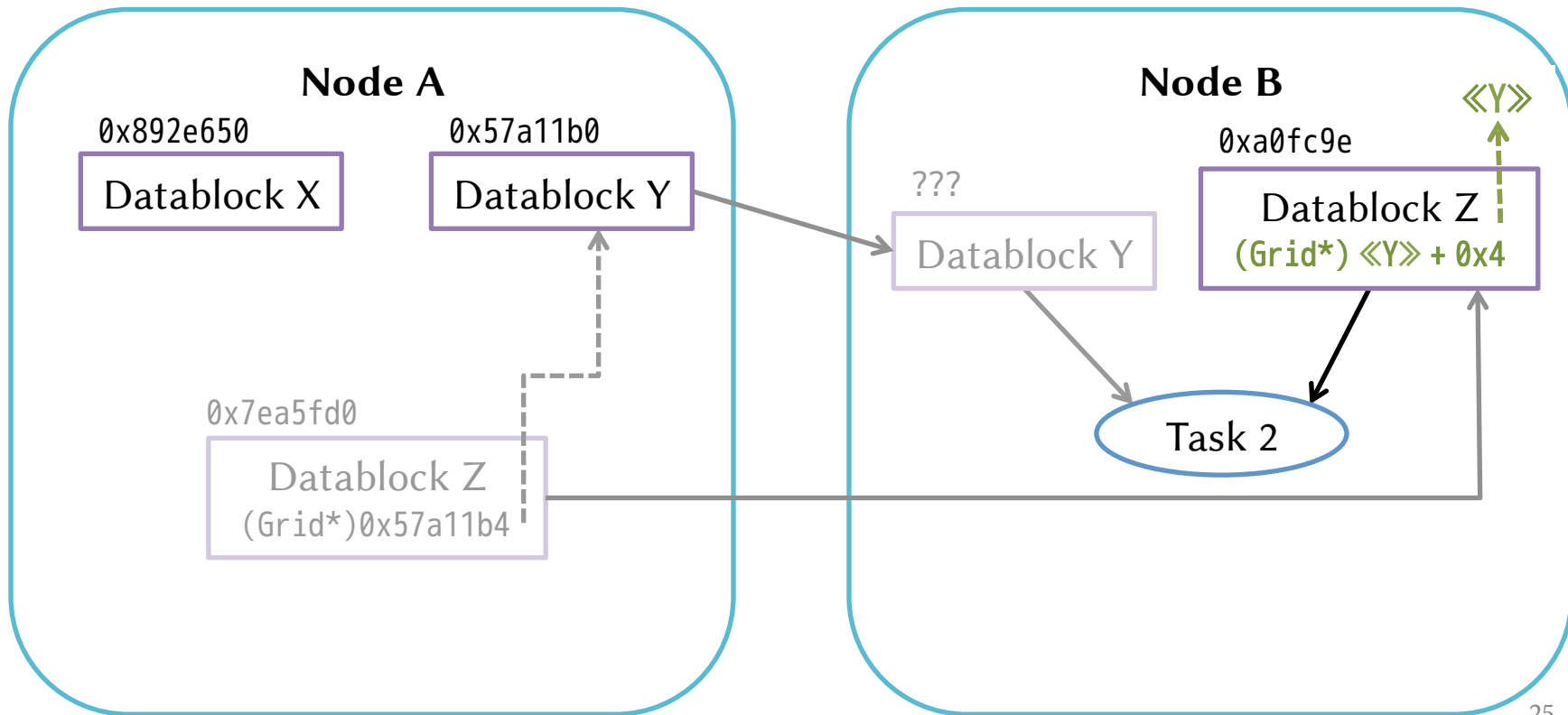
Problem #1: Example (revisited)



Proposed Solution #1

Sanitize all objects that are persisted in datablocks across multiple tasks, using position-independent C++ “pointer” objects.

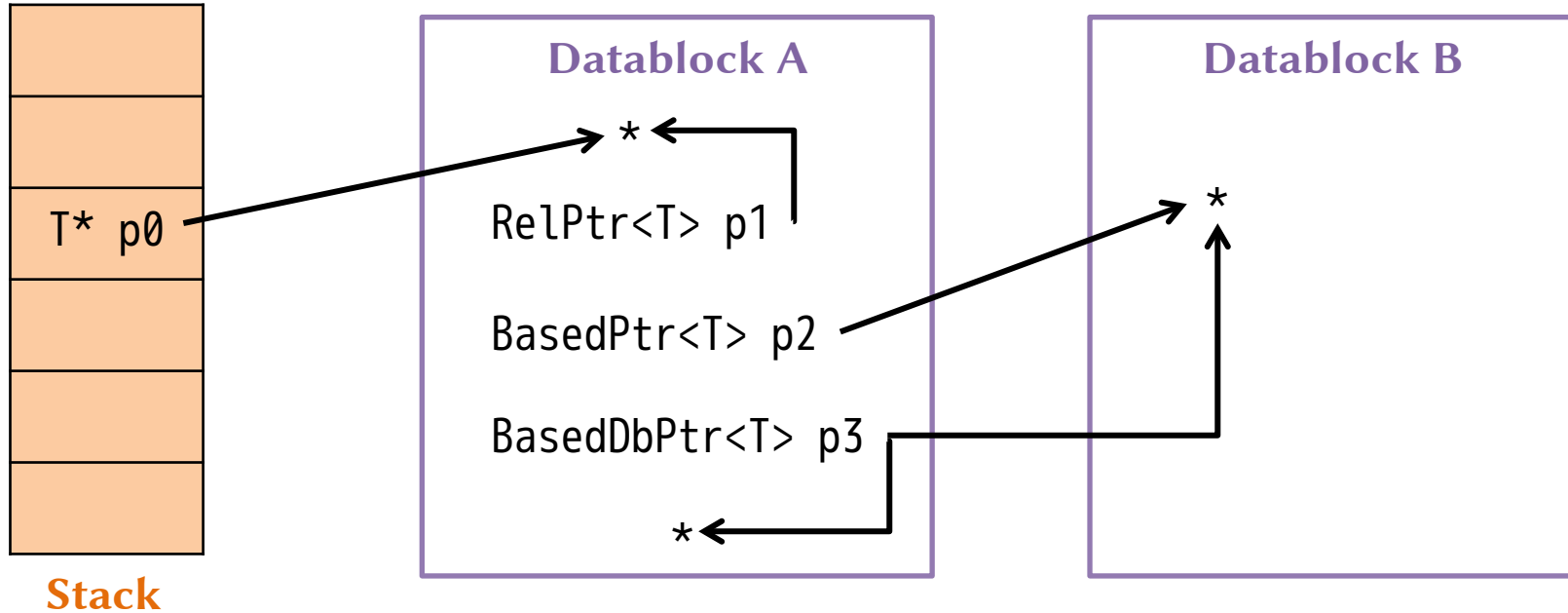
Example using Based Pointer



ocxxr: C++ Support for OCR

- Provides C++ API support for:
 - Initializing C++ objects within datablocks
 - Datablocks as allocation arenas
 - Position-independent pointer objects
- Additional benefits:
 - Improved static type checking (via templates)
 - Little-to-no overhead from inlined C++ wrappers
- Comprises 1275 Logical SLOC

Classes of Pointers in ocxxr



ocxxr: API Example

```
using namespace ocxxr;
```

```
struct Node {  
    int value;  
    RelPtr<Node> left;  
    RelPtr<Node> right;  
};
```

```
struct Tree {  
    RelPtr<Node> root;  
    // ... methods ...  
};
```

```
void SubTask(int i, Arena<Tree> tree) {  
    Node *tree_root = tree->root;  
    if (i < 10) {  
        // ... do something with tree_root ...  
        TaskBuilder<decltype(SubTask)> builder = /* ... */;  
        builder.CreateTask(i+1, tree);  
    } else { Shutdown(); }  
}  
  
void MainTask() {  
    Arena<Tree> tree = Arena<Tree>::Create(ARENA_SIZE);  
    // ... set up tree ...  
    TaskBuilder<decltype(SubTask)> builder = /* ... */;  
    builder.CreateTask(0, tree);  
}
```

Bonus: ocxxr Expressiveness

Fib in ocxxr: 63 LSLOC

```
void FibContinuation(
    ocxxr::Event<u32> &output,
    ocxxr::Datablock<u32> lhs,
    ocxxr::Datablock<u32> rhs) {
    // left_value + right_value -> output
    lhs.data() += rhs.data();
    rhs.handle().Destroy();
    output.Satisfy(lhs); // type-checked
}
```

Fib in OCR: 132 LSLOC

```
ocrGuid_t fib_continuation(
    u32 paramc, u32* paramv,
    u32 depc, ocrEdtDep_t depv[]) {
    // unpacking arguments
    ocrGuid_t output = *(ocrGuid_t*)paramv;
    u32 *lhs = depv[0].ptr;
    u32 *rhs = depv[1].ptr;
    // left_value + right_value -> output
    *lhs += *rhs;
    ocrDbDestroy(depv[1].guid);
    ocrEventSatisfy(output, depv[1].guid);
    return NULL_GUID; // unused return value
}
```

Pointer Conversion Algorithm

- Two main phases:
 1. Identify all types persisted in datablocks
 2. Process class types to convert pointer fields
- Template instances handled individually
- Prototyped using Clang LibTooling

Pointer Conversion Example

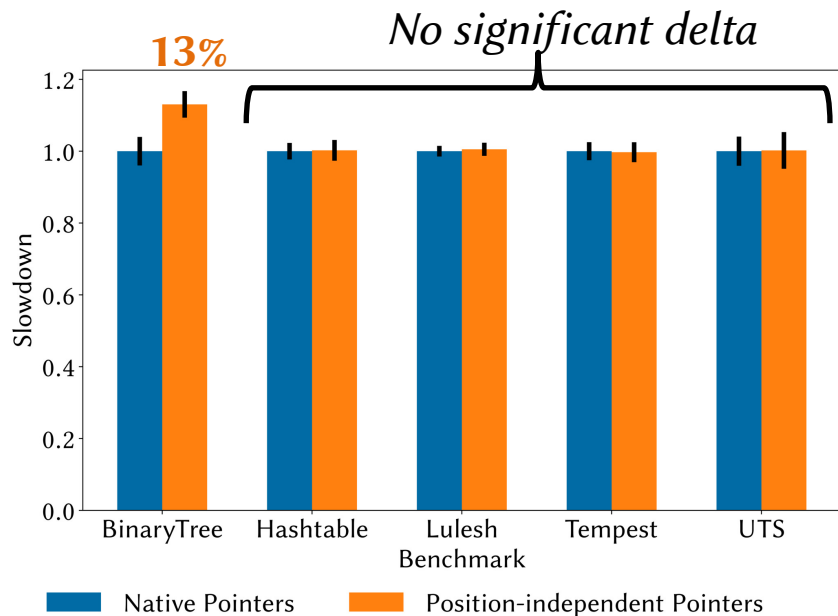
```
struct MyNode {  
    double value;  
    ocxxr::BasedDbPtr<MyNode> next;  
};  
  
void MyTask(ocxxr::Arena<MyNode> arena) {  
    MyNode *head = arena.data_ptr();  
    MyNode *next = arena.New<MyNode>();  
    next->value = 1234.56;  
    // the pointer and its addressee  
    // are within the same datablock  
    head->next = next;  
}
```

1. Identify types τ in task dependence inputs
2. Find all pointer members of the class type τ
3. Replace pointer types with BasedDbPtr types
4. Recursively fix pointers in BasedDbPtr target types

Algorithm Limitations

- Does not find types “hidden” by casts
`void* → SomeType*`
- Does not handle C++ STL classes (allocators)
- May transform classes used in temporary data
- Currently lacks alias analysis: cannot identify safe candidates for representation as `RelPtr<T>`

Slowdown vs Native Pointers

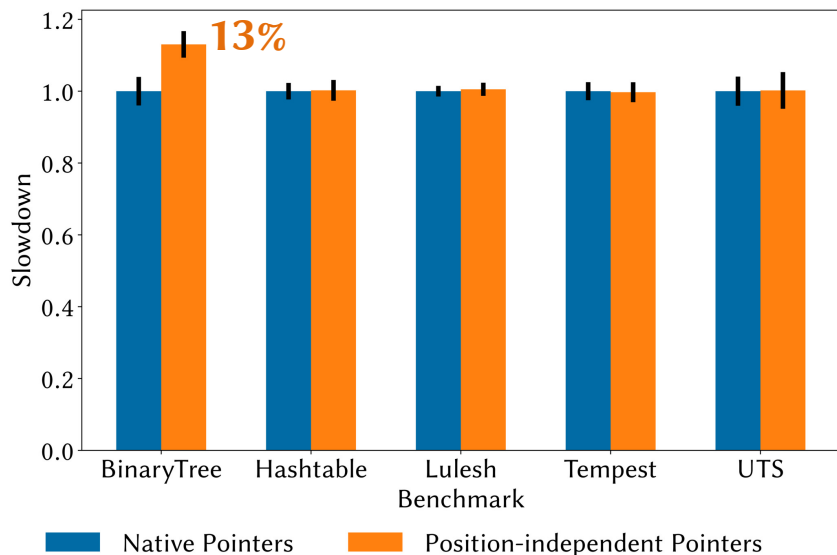


Benchmark	Position-independent
BinaryTree	1.130 ± 0.037
HashTable	1.002 ± 0.029
LULESH	1.005 ± 0.018
Tempest	0.997 ± 0.028
UTS	1.002 ± 0.051

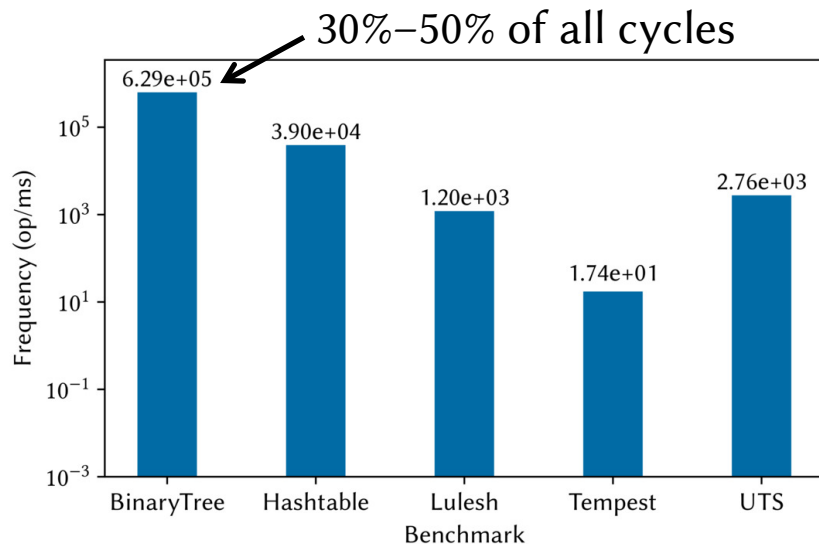
Error margins give 95% confidence interval

Slowdown ~ Pointer Operations

Slowdown vs Native Pointers

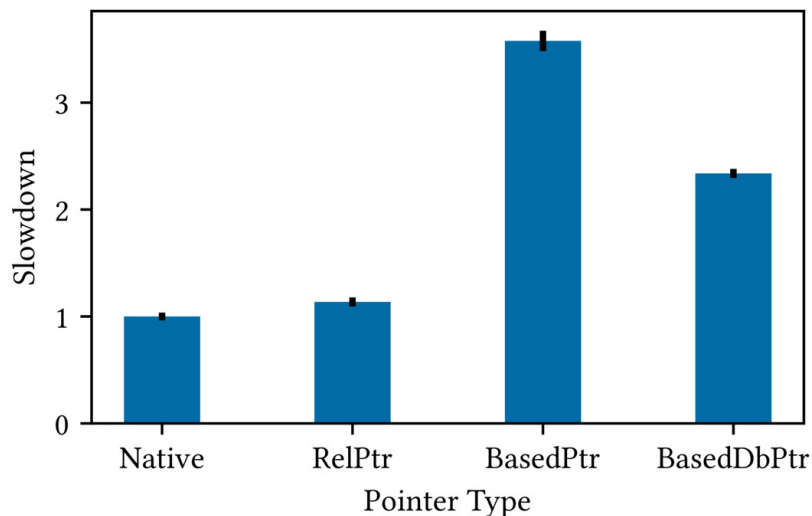


Pointer Operation Densities



BinaryTree Benchmark

Slowdowns for Pointer Variants



Variant	Slowdown
T^* (<i>native</i>)	$1.000 \pm .036$
RelPtr<T>	$1.136 \pm .042$
BasedPtr<T>	$3.578 \pm .095$
BasedDbPtr<T>	$2.338 \pm .042$

Error margins give 95% confidence interval

Summary of Resolution for #1

- ocxxr library: C++ support for OCR
 - Safe pointer-object encoding in datablocks
 - Enabled porting C++ framework to OCR
 - Marginal overhead measured in real kernels
- Conservative pointer-conversion algorithm with Clang-based transformation tool

Outline

1. Position-independent object encoding for migratable datablocks
2. Practical support for blocking constructs in lightweight tasking runtimes
3. CnC-OCR: a productivity layer for OCR
4. Conclusions and future directions

Problem #2: Description

- Default scheduling strategy in OCR and HClib does not properly support blocking
- The *runtime* introduces new deadlock scenarios through unsafe optimizations
- How do we support blocking constructs safely and efficiently?

Examples of Blocking Constructs

Habanero-C

- `Finish scope`
Blocks until all *async* tasks within the finish scope complete
- `Future.wait()`
Blocks until the target future task has completed

OCR

- `ocrWait(event)`
Blocks until *event* is triggered
- Remote datablock create
- Remote task create
- Remove event satisfy
- ...

Problem #2: Example

// This code executes on Worker-A
➡ `auto f0 = hclib::async_future([]() {
 /* . . . */ });`
// future-task f0 is stolen by Worker-B

➡ `auto f1 = hclib::async_future([]() {
 return f0.wait(); });`
// future-task f1 is stolen by Worker-C

➡ `t2: hclib::async([]() { f1.wait(); });`
*// above task stolen by Worker-C
// Worker-C blocks on f0.wait()
// and starts looking for more work
// (DEADLOCK!)*

Worker-A	Worker-B	Worker-C
async f0	steal f0	
async f1	running f0	steal f1
async t2	...	running f1
...		block on f0
		"help"
		steal t2
		block on f1
		...

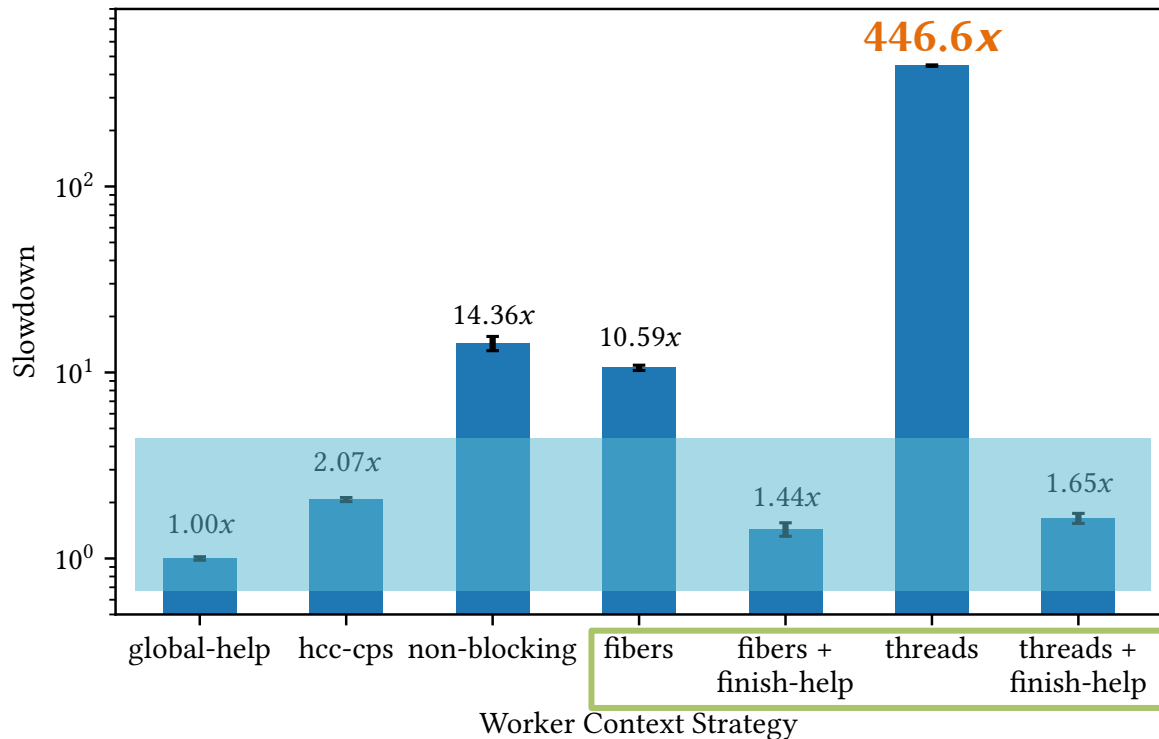
Proposed Solution #2

- Identify the minimal support needed for correct scheduling of blocking constructs in OCR, Habanero-C, or similar runtime
- Select a range of possible solutions
- Evaluate selected solutions for performance and programmability tradeoffs

Proposed Solution #2

- Identify the minimal support needed for correct scheduling of blocking constructs in OCR, Habanero-C, or similar runtime
- Select a range of possible solutions
- Evaluate selected solutions for performance and programmability tradeoffs
- See thesis text for full details

Strategy Overheads: Fibonacci



Strategy Recommendations

- Use Threads + Finish-Help for development
- Use Fibers + Finish-Help in production
- Use CPS-transform (compiled or manual) for failure-prone production environments

Summary of Resolution for #2

- Identified previously-undiscovered deadlock scenarios in OCR and Habanero-C
- Implemented and evaluated several safe strategies to handle blocked tasks
- Added novel, safe Finish-Helping optimization
- Defined guidelines for applying the strategies

Outline

1. Position-independent object encoding for migratable datablocks
2. Practical support for blocking constructs in lightweight tasking runtimes
3. CnC-OCR: a productivity layer for OCR
4. Conclusions and future directions

OCR: Intended Users

- Concurrency experts / “hero programmers”
- Library implementers
- Compiler back-end engineers



Note: *mere mortals* are not the target users.

Problem #3: Description

- OCR is designed as a low-level API
- Higher-level abstractions on top of OCR is intended to improve the OCR API
- Existing higher-level languages for OCR do not stay true to the OCR data model
(Hierarchically Tiled Arrays, HCLib)

Proposed Solution #3

CnC for the Open Community Runtime (OCR):

- increase productivity
- simplify tuning
- support explicit hierarchy

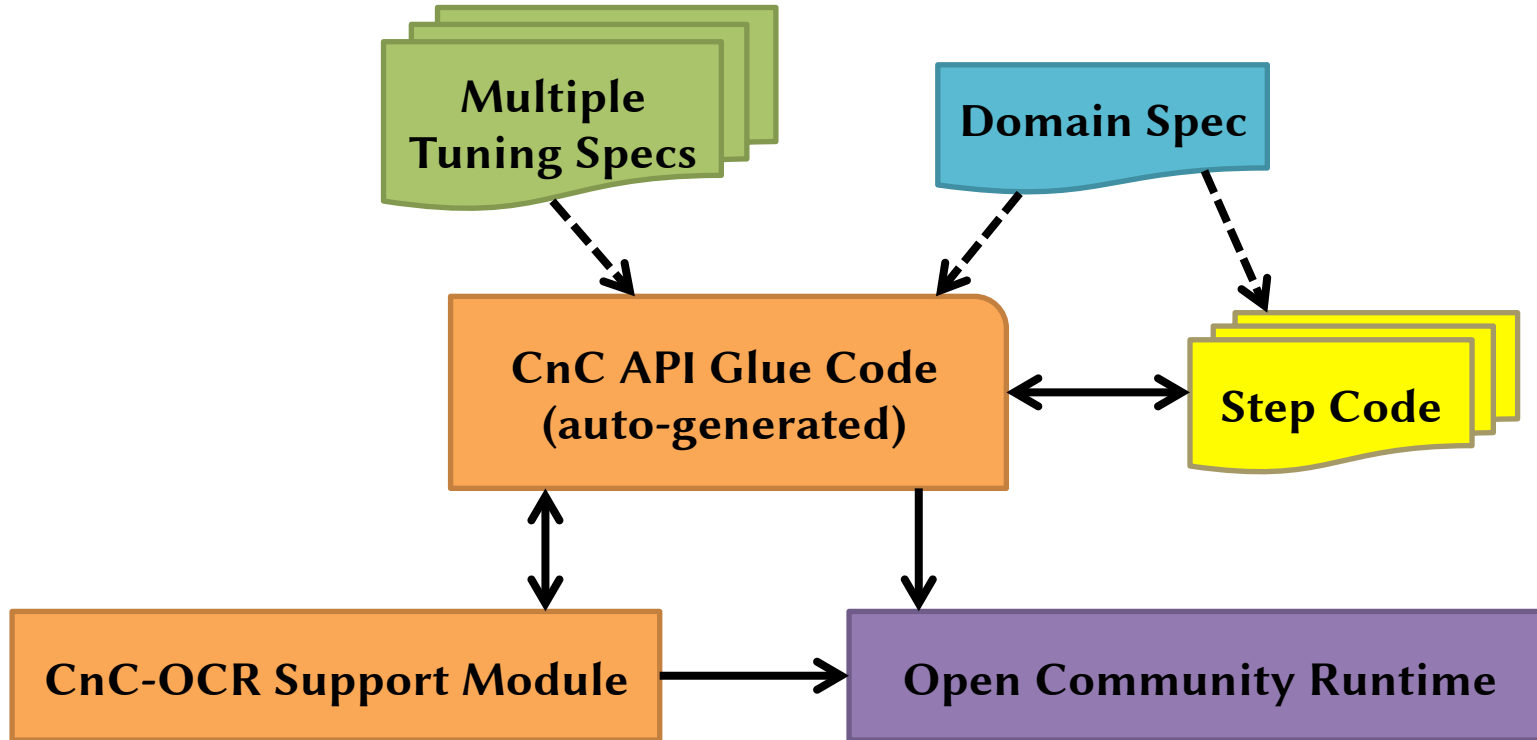
CnC Programming Model: Dependence Programming

- Problem partitioned into *steps* and *items*
- Steps/items partitioned into *collections*
- Step/item instances have unique *tags*
- Dependencies specified relative to tags

CnC Programming Model: Separation of Concerns

- High-level dependence specification
- Individual compute step implementations
- Platform-specific tuning specification

CnC-OCR Software Architecture



CnC Example: Smith-Waterman Scoring Matrix

	--	A	G	C	A
--	0	-1	-2	-3	-4
A	-1	2	1	0	-1
C	-2	1	0	3	2
A	-3	0	-1	2	5
C	-4	-1	-2	1	4
A	-5	-2	-3	0	3

The diagram illustrates the optimal path for sequence alignment using a Smith-Waterman scoring matrix. The path is marked by arrows, starting from the top-left cell (0,0) and ending at the bottom-right cell (6,5), which is circled in red. The path follows the sequence: (0,0) → (1,1) → (2,2) → (3,3) → (4,4) → (5,5) → (6,5).

CnC Example: Smith-Waterman

Domain Specification

```
[ int above[#tw] : i, j ];  
[ int left[#th]  : i, j ];  
[ SeqData *data : () ];
```

item (data) collection declarations

```
( swStep: i, j )
```

step collection declaration

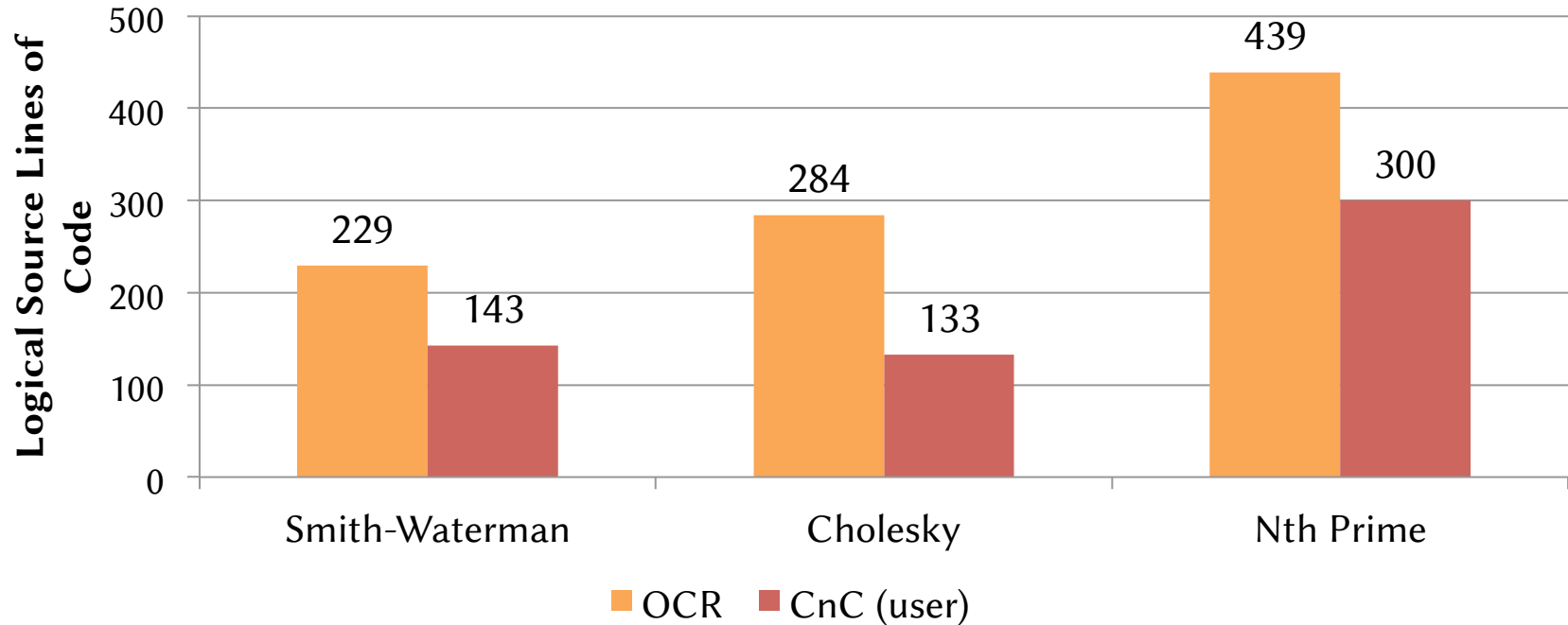
```
<- [ data: () ],  
    [ above: i, j ] $when(i > 0),  
    [ left: i, j ]  $when(j > 0)
```

step-input item dependence relations

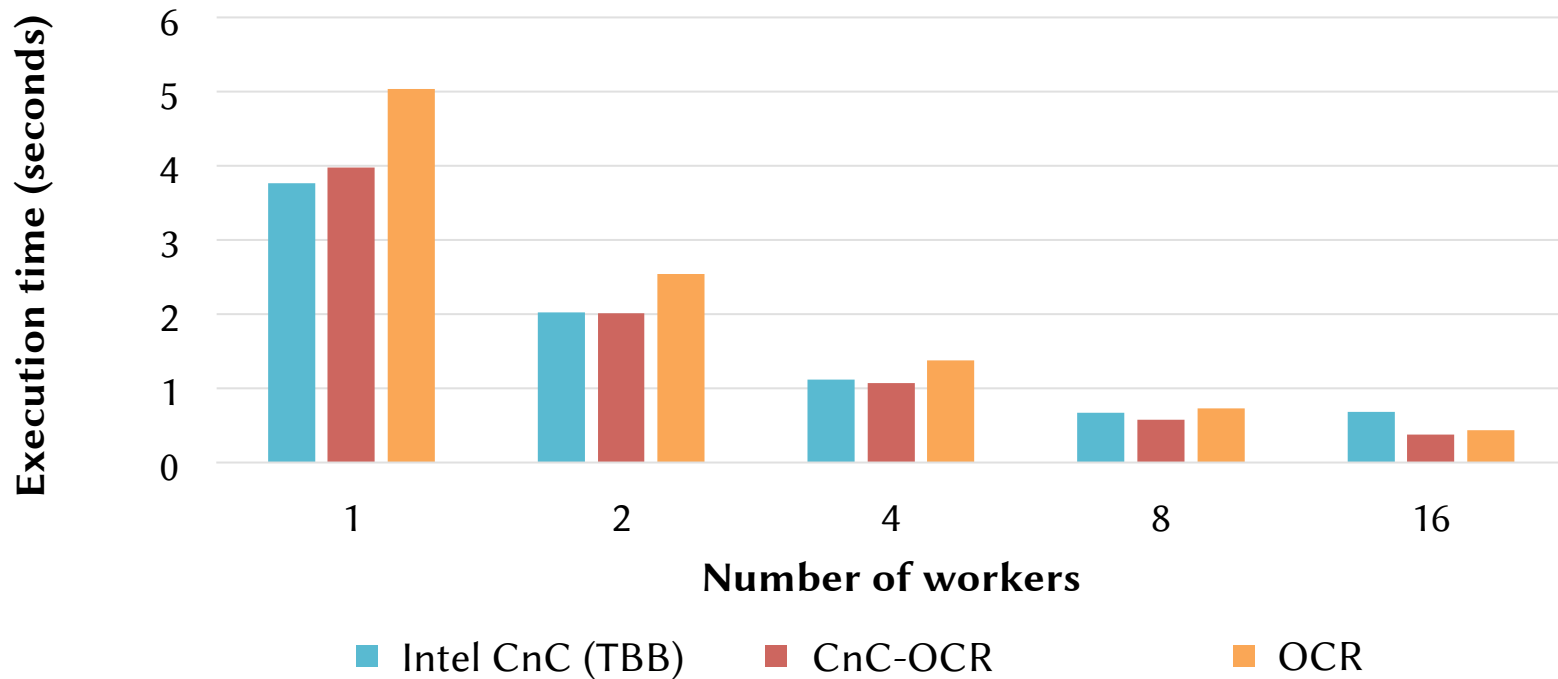
```
-> [ below @ above: i+1, j ],  
    [ right @ left: i, j+1 ],  
    ( swStep: i+1, j )  
      $when(i+1 < #nth);
```

step-output relations (items and steps)

CnC Productivity Results



CnC Cholesky Performance Results



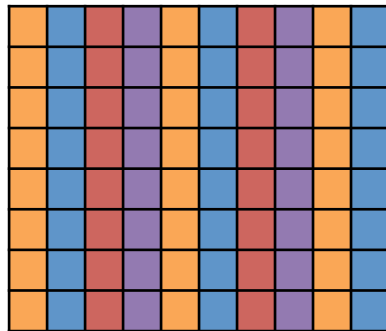
CnC Example: Smith-Waterman

Tuning Specification

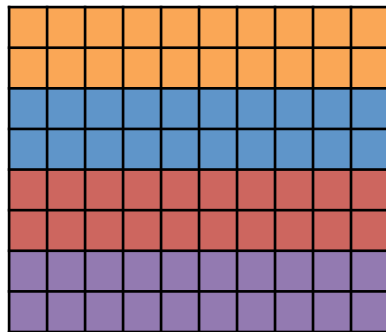
```
[ above ]: {  
    distfn: (i / 16) % $RANKS  
};
```

```
[ left ]: {  
    distfn: (i / 16) % $RANKS  
};
```

```
( swStep ): {  
    placeWith: above  
};
```

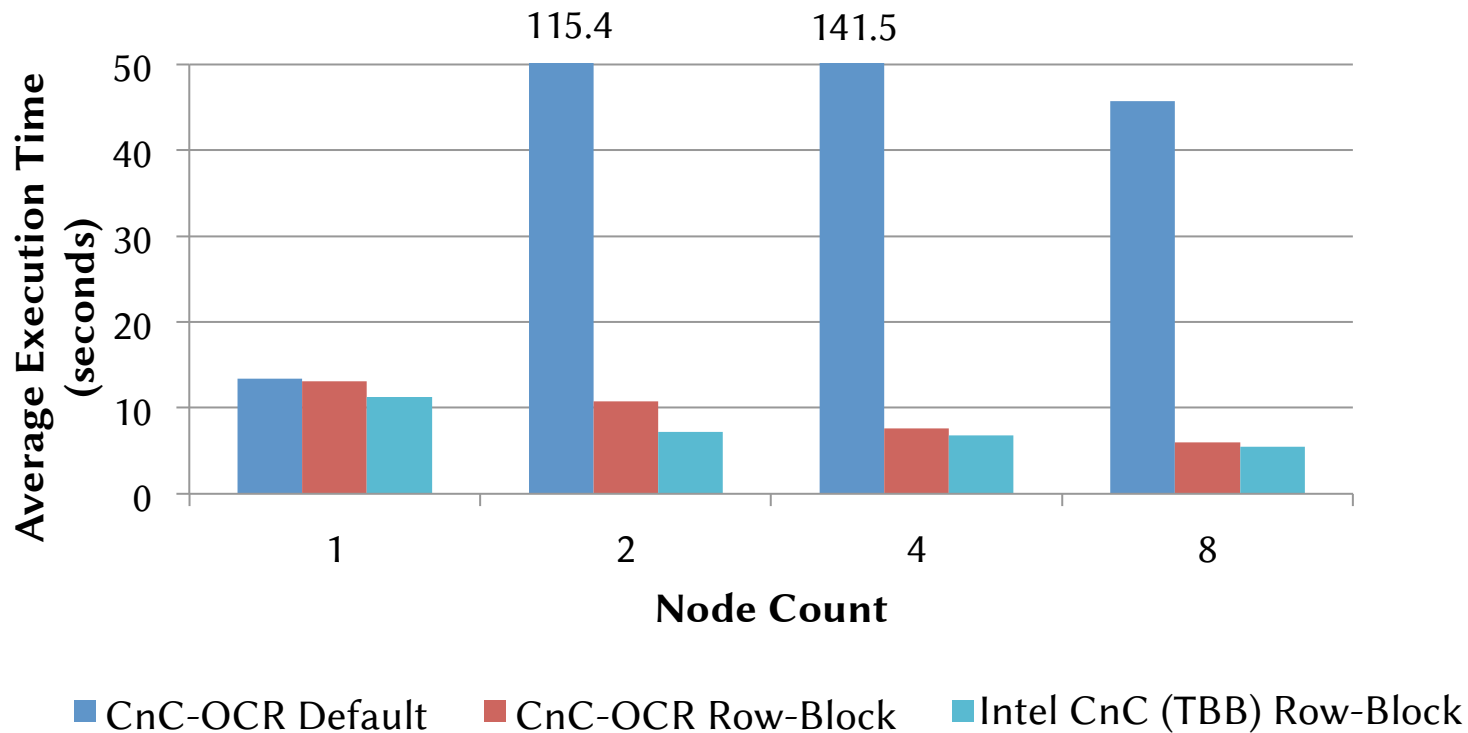


Default distribution
(cyclic on last tag
component: column)

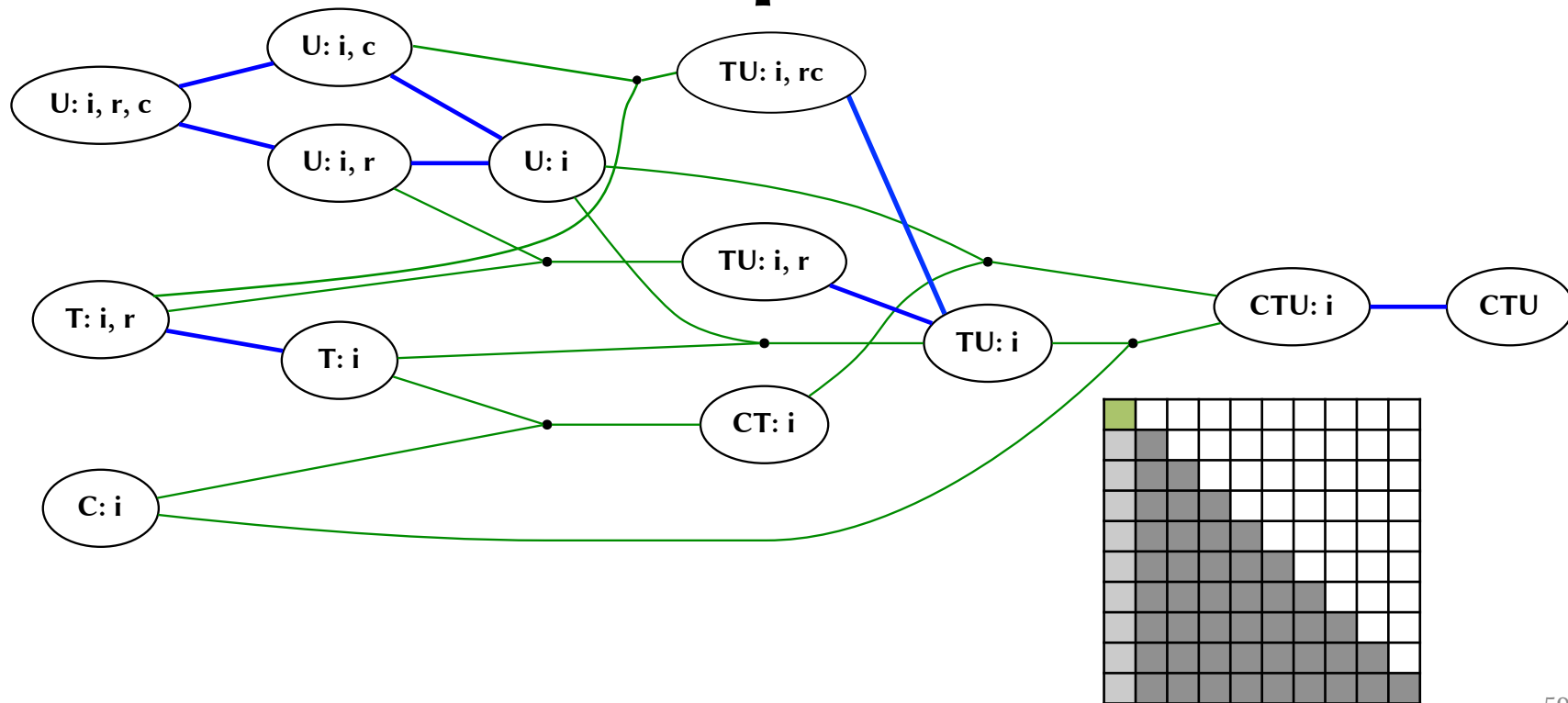


Customized
row-block distribution

CnC Smith-Waterman Distributed Tuning Results



Cholesky Tuned w/CnC Hierarchy: Lattice of Step Granularities



Habanero CnC-Framework

- Over 3300 Logical SLOC
 - C/C++: 2558 Logical SLOC
 - Python: 818 Logical SLOC
- Designed for extensibility
 - Shared parser code for *domain* and *tuning* DSLs
 - Supports both Intel CnC and OCR as back-ends
 - Forked to target other runtimes (HCMPI, HPX-5)
- Tools for debugging and software hierarchy

External Research with the Habanero CnC-Framework

- *Communication-avoiding ray tracing for exascale computing*
Ellen Porter, Washington State University, Master's Thesis
- *Improving programmability and performance for scientific applications*
Chenyang Liu, Purdue University, PhD Thesis
- *Programming HPX-5 with Concurrent Collections*
Buddhika Chamith, Indiana University,
PhD Student Poster at SC15

Summary of Resolution for #3

- Integration of CnC model with OCR
- Extensible toolchain and DSL design
 - Support for multiple runtime back-ends
 - Easily adaptable to new research projects
- Novel DSL for performance tuning
- First in-depth study of CnC Hierarchy concepts

Outline

1. Position-independent object encoding for migratable datablocks
2. Practical support for blocking constructs in lightweight tasking runtimes
3. CnC-OCR: a productivity layer for OCR
4. Conclusions and future directions

Summary

Addressed several runtime challenges for extreme-scale by improving task and data abstractions:

1. Improved programmability & safety of C++ Tempest applications on OCR via custom pointer objects
2. Improved liveness guarantees & performance for scheduling tasks with blocking constructs
3. Improved productivity & performance for applications on OCR via the CnC-OCR toolchain

Conclusions

- Higher-level languages and libraries are a critical component of a runtime ecosystem
- Alternative implementations help avoid forced choice between productivity and performance
- Separation of concerns can improve productivity and facilitate performance tuning

Future Work

- Compiler / tools support for OCR development:
 - Datablock alias analysis
 - Automatic data partitioning
 - Automatic CPS transformation of blocking code
- Static checks for OCR applications:
 - Data-race detection for OCR applications
 - Safety determination for global-help scheduling
- Higher-level programming models for OCR:
 - Legion / Realm on OCR
 - Chapel on OCR

Acknowledgements

- *Committee*: Thanks for your time and input!
- *Habanero Team*: It's been great working with you!
- *Intel X-Stack Team*: Thanks for your mentorship and support! I really enjoyed working with all of you!

This material is based upon work supported by the Department of Energy, Office of Science, under Award Number DE-SC0008717.

Runtime Feature Comparison

	<i>Pointer Safety</i>	<i>Blocking</i>	<i>High-level languages</i>
Charm++	Programmer's responsibility	ucontext	Not yet supported
HPX	Programmer's responsibility	ucontext or boost::context	Not yet supported
OCR	Static and dynamic checks	Many options presented	CnC
Realm	Programmer's responsibility	ucontext	Legion, Regent
UPC++	No data migration	SPMD blocking	Not yet supported

Backup Slides

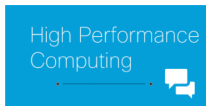
Related Publications

- *A Marshallled Data Format for Pointers in Relocatable Data Blocks.*
Nick Vrvilo, Lechen Yu and Vivek Sarkar.
The 2017 International Symposium on Memory Management (ISMM). June 2017.
- *Declarative Tuning for Locality in Parallel Programs.*
Sanjay Chatterjee, Nick Vrvilo, Zoran Budimlic, Kathleen Knobe, Vivek Sarkar.
The 45th International Conference on Parallel Processing (ICPP), August 2016.
- The CnC work has been presented at multiple CnC Workshops and at multiple Intel X-Stack Traleika Glacier Project Workshops.

Note: Counting Lines of Code

- USC Universal Code Count (UCC)
 - Contracted by Department of Defense
 - Standard, open-source tool for estimating the effort to create a software project
- Logical Source Lines of Code (LSLOC)
 - Primary metric of UCC
 - Style-agnostic measurement of lines of code


Similar Problems for One-sided Comm & C++ Objects



High Performance Computing

High Performance Computing Networking

How to send C++ STL objects in MPI?



Jeff Squires - January 24, 2012 - 7 Comments

A while ago, [Brock Palen](#) [tweeted me an MPI question](#): how does one send Standard Template Library (STL) C++ objects in MPI?

The problem that Brock is asking about is that STL objects tend to be variable size and type. The whole point of the STL is to create flexible, easy-to-use "containers" of arbitrary types. For example, STL lists allow you to create an arbitrary length list of a given type.

To cite a concrete example, let's say that my application has an STL vector object named `my_vector`

In an effort to keep conversations fresh, Cisco Blogs closes comments after 60 days. Please visit the [Cisco Blogs hub page](#) for the latest content.


7 Comments



Jeff Hammond
January 24, 2012 at 12:24 pm

How does this work with one-sided, where remote unpacking must be done inside of MPI, rather than in the glue code between Boost::MPI and the MPI C bindings? It would be good to show potential users how to use MPI datatypes in the official API to communicate STL objects, rather than to punt this problem to "magical" Boost and its antiquated MPI support.

0 likes



Jeff Squires
January 24, 2012 at 12:44 pm

boost.mpi doesn't work with one-sided. boost.mpi only "mostly" supports MPI-1.1*. Since the MPI bindings are in C, they don't have good native support for things like the STL. That's why boost.mpi exists. I agree that its support is a bit antiquated at this point; it would be great if someone could update it to include a bit more than "most of MPI-1.1". Additionally, such high-level concepts tend to not map well to hardware-offload scenarios. Meaning that one-sided operations of STL-like containers won't map well to hardware-accelerated message passing solutions because the hardware won't understand the (potentially) complex run-time created data layouts -- such code paths will likely fail

Q: How do you transmit C++ objects using MPI?

A: Use the great Boost.MPI and Boost.Serialize libraries!

Comment: Cool! Does this do one-sided communication too?

Reply: No, that doesn't work...

Classes of Pointers in ocxxr

Pointer Variant	Address Computation	Use Case
T* (<i>native</i>)	<code>*this</code>	Temporary (non-persisting) data.
RelPtr<T>	<code>*(this + offset)</code>	Pointer and target must be located within the same datablock. (I.e., the relative offset never changes.)
BasedPtr<T>	<code>*(base() + offset)</code> <code>base() ==> &block(id)</code>	Pointer and target probably not located within the same datablock.
BasedDbPtr<T> (<i>hybrid</i>)	<code>if (id.IsValid())</code> <code>*(base() + offset)</code> <code>else</code> <code>*(this + offset)</code>	Pointer and target sometimes in the same datablock, sometimes not. Has extra overhead for doing pointer's datablock lookup on assignment.

Experimental Setup

Software & Benchmarking

- Ubuntu 16.04 LTS (Xenial)
- Clang v3.8
- 100 runs per configuration
- 95% confidence intervals
- Native pointers as baseline
(can't run distributed)

Hardware

- Single-node system
(but also works distributed)
- 3.50GHz Intel Core i7
Ivy Bridge 4-core CPU
- Turbo boost disabled
- 8GiB DDR3

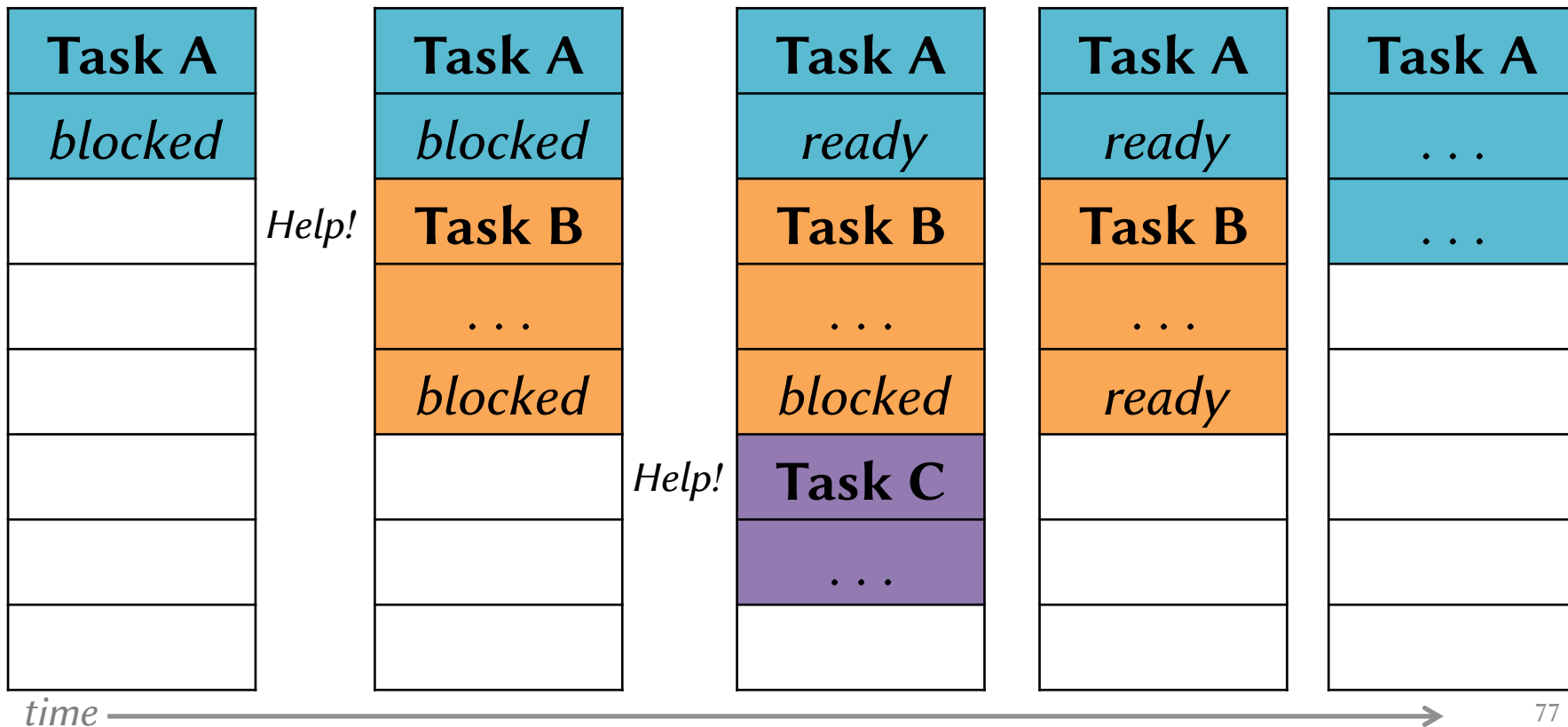
Benchmarks

Benchmark	Purpose	Description
<i>BinaryTree</i>	RelPtr-heavy	Builds a binary search tree data structure within a single arena datablock, performing a large number of put/get ops.
<i>HashTable</i>	BasedPtr-heavy	Builds a hashtable, with buckets of entries distributed across multiple datablocks. Performs a large number of put/get ops, acquiring buckets on-demand.
<i>LULESH</i>	Mesh access	Port of LULESH 2.0 code to ocxxr (based on CnC-OCR port).
<i>Tempest</i>	Motivating application	Small kernel using the Tempest climate-simulation framework, modeling a patch on the cube-sphere grid.
<i>UTS</i>	BasedDbPtr-heavy	Builds the tree generated by the “Unbalanced Tree Search” benchmark across many datablocks.

Habanero Programming Model

- Hierarchy of concurrency constructs
 - Increasing expressiveness with more constructs
 - Better safety guarantees with restricted subset
 - Has *async/finish* model (X10) at the core
 - Extended with futures, data driven tasks and promises
- Several implementations exist:
 - JVM: Habanero Java, HJ-lib, Habanero Scala
 - C/C++: Habanero-C, HClib

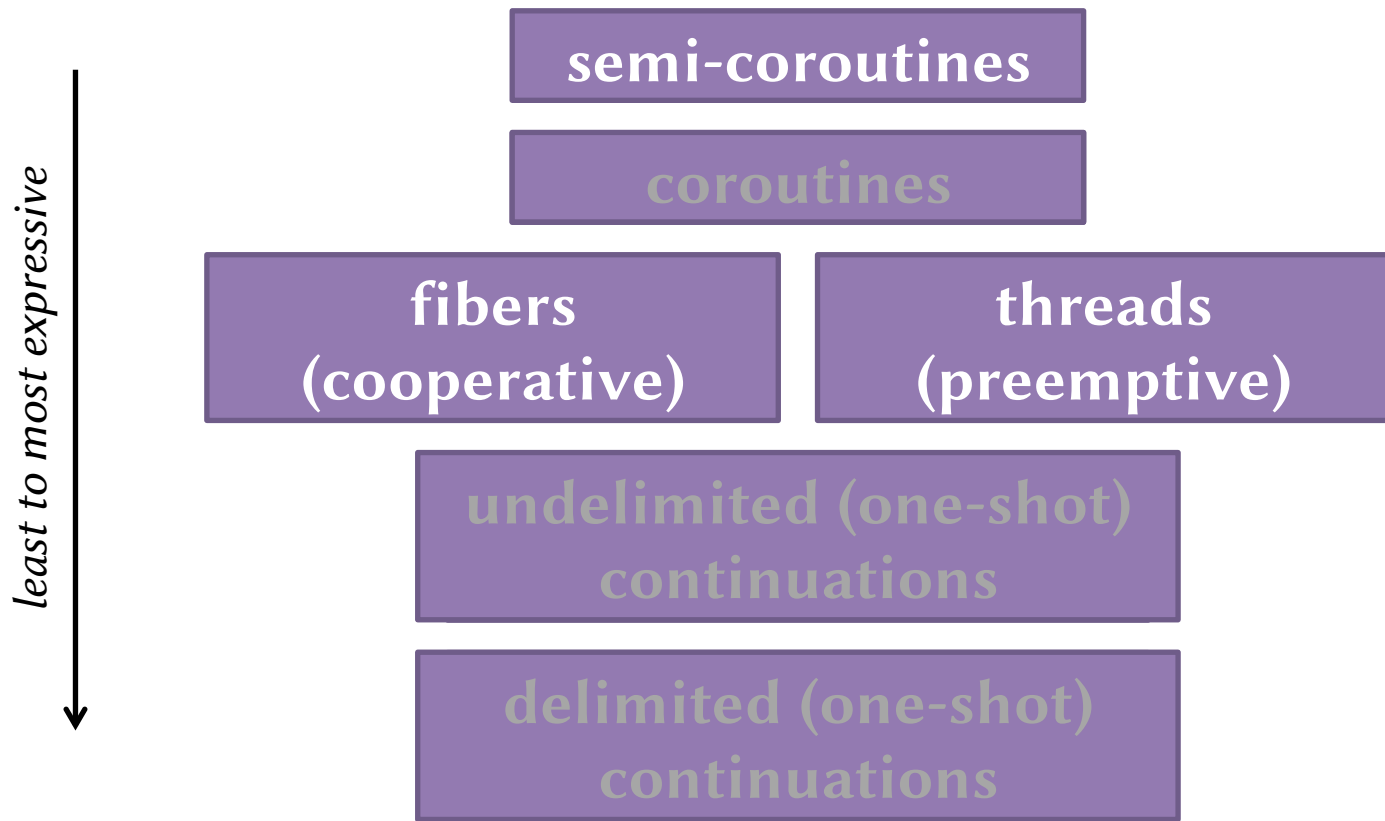
Thread Stack + Global Helping



Global-Helping and Deadlocks

The Global-Helping optimization can create new deadlock scenarios if and only if it is possible to create a dependence from a later blocking task to an earlier blocking task.

Blocking-support Options



Selected Solutions

- **Compensation with Threads:**

Create a new OS thread each time a worker blocks.

- **Compensation with Fibers:**

Create a new fiber to each time a worker blocks.

- **Transform blocking tasks into Semi-coroutines:**

Save task's continuation when blocking (compiler supported).

- **Rewrite with Non-blocking constructs:**

Application is written in a fully non-blocking style, using chained *futures* to handle all synchronization (manual CPS transform).

This makes heavy use of *futures* with `async_await`.

Additional Optimized Variants

- **Threads + Finish-Helping:**

- Based on *Threads*, but adds a provably-safe “helping” optimization to reduce compensation threads
- The Finish-Helping optimization restricts “helping” targets to only tasks in the current finish scope

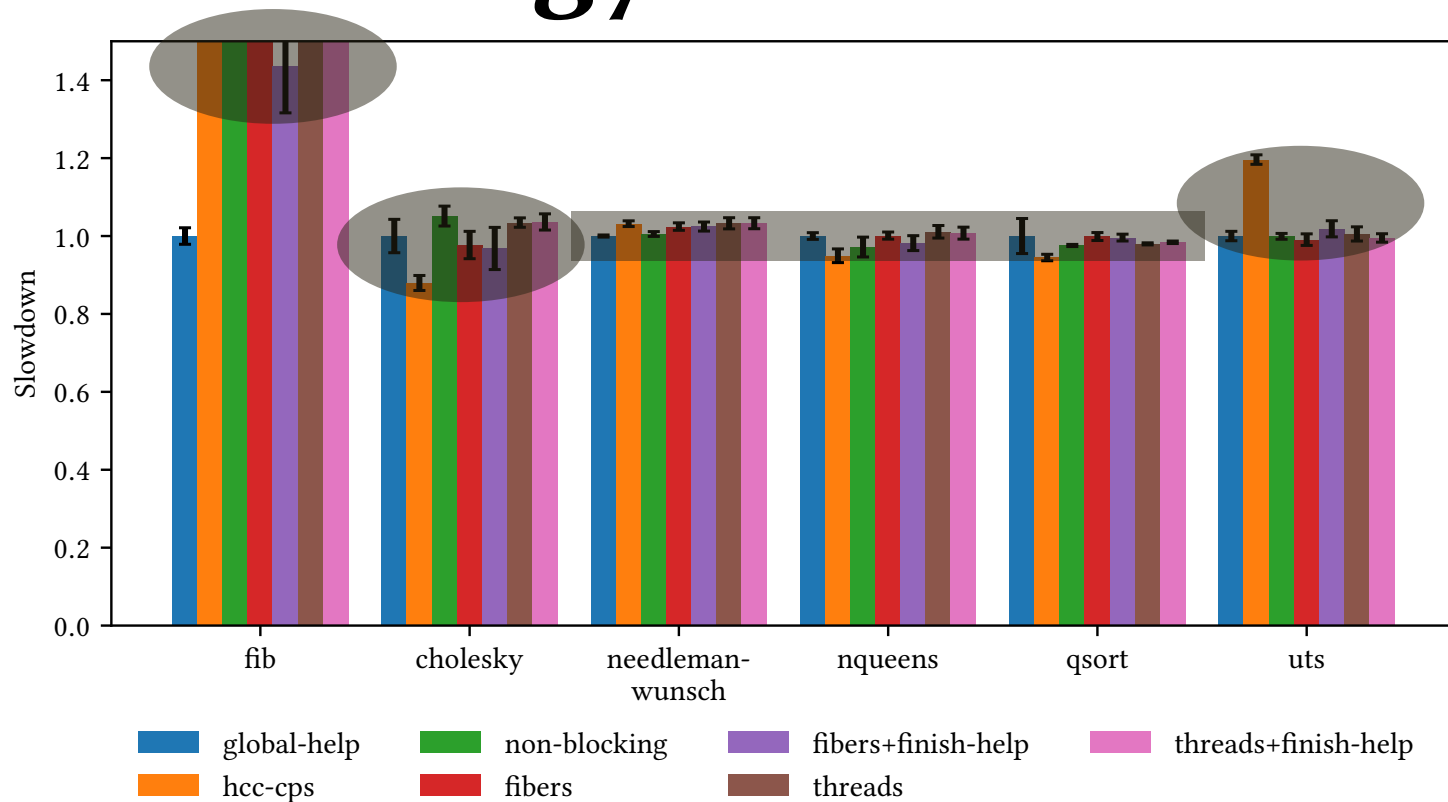
- **Fibers + Finish-Helping:**

Similar to *Threads + Finish-Helping*, but using *fibers* rather than OS threads

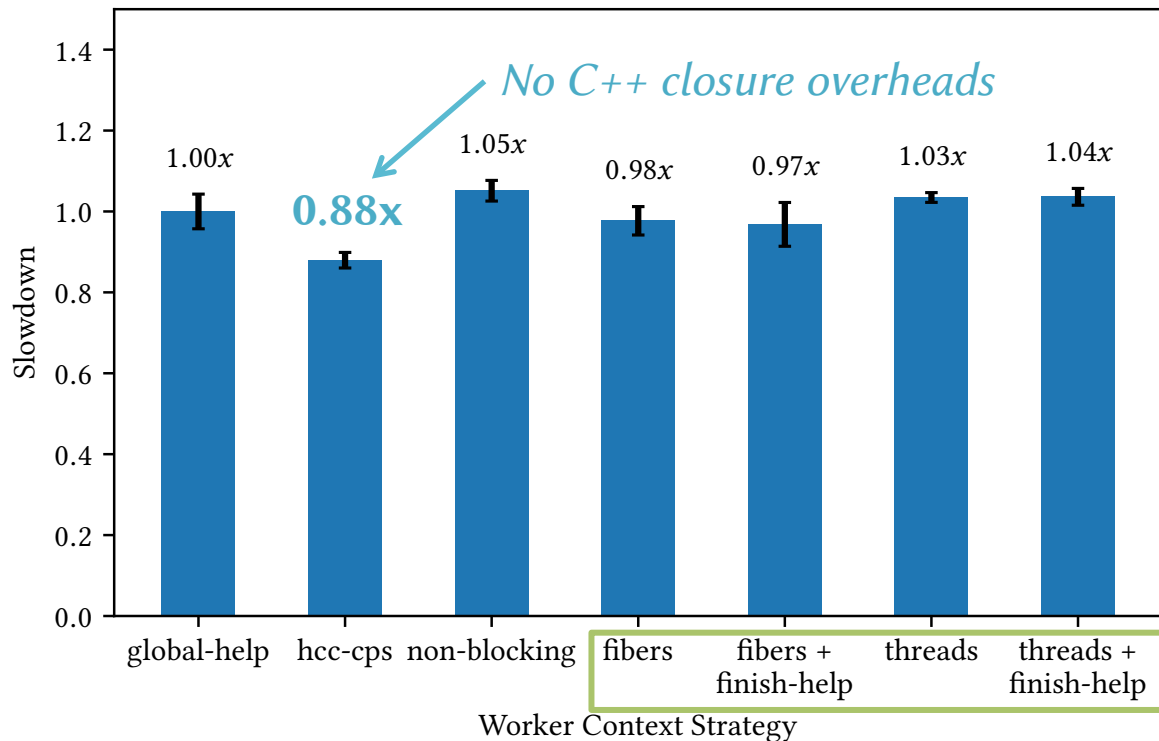
Proofs for Worker Strategies

- Global-Helping is safe for *async/finish*
- Precise conditions where Global-Helping-induced deadlocks can occur
- Finish-Helping is safe for all constructs
- Deadlock-freedom for *async/finish + futures* requires *semi-coroutines* (or equivalent)

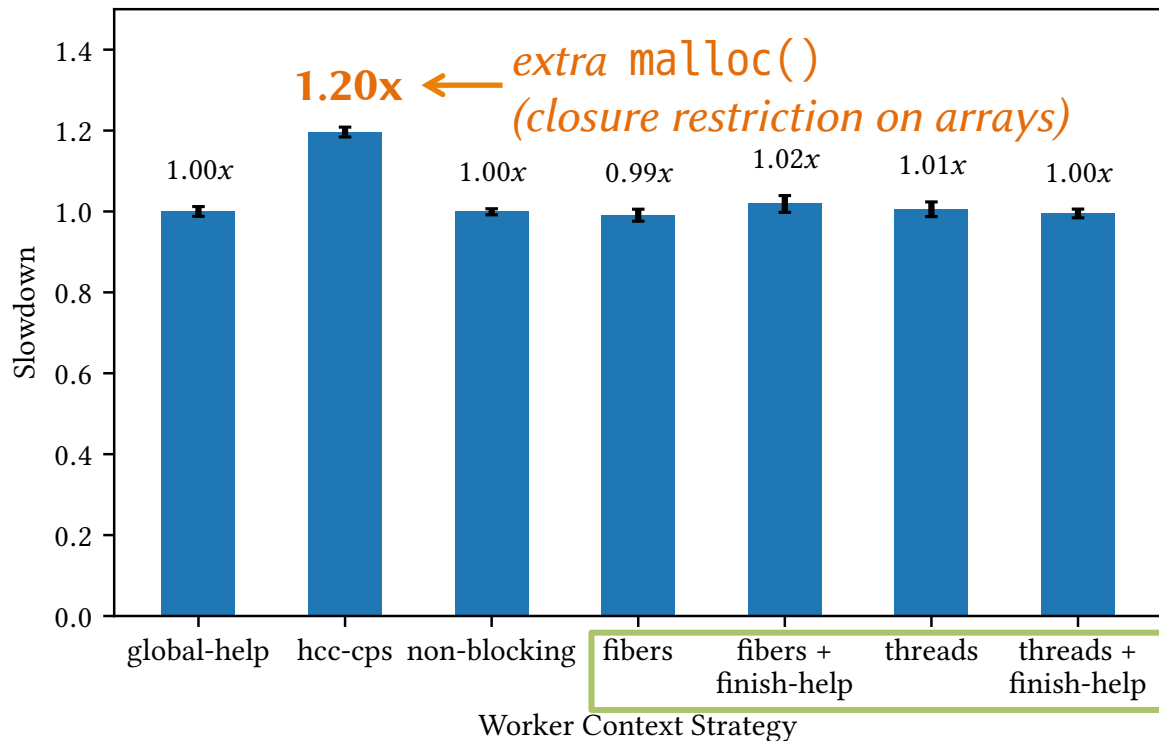
Strategy Overheads



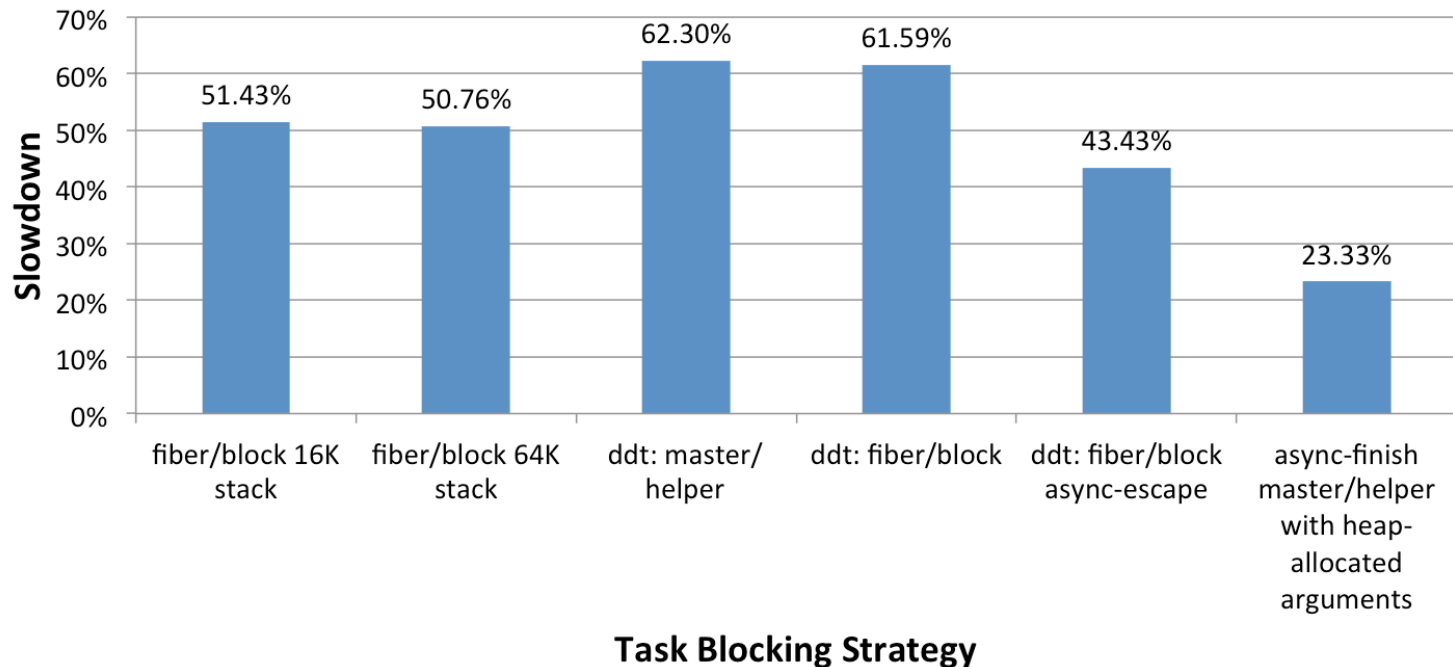
Strategy Overheads: Cholesky



Strategy Overheads: UTS



Initial Performance Results



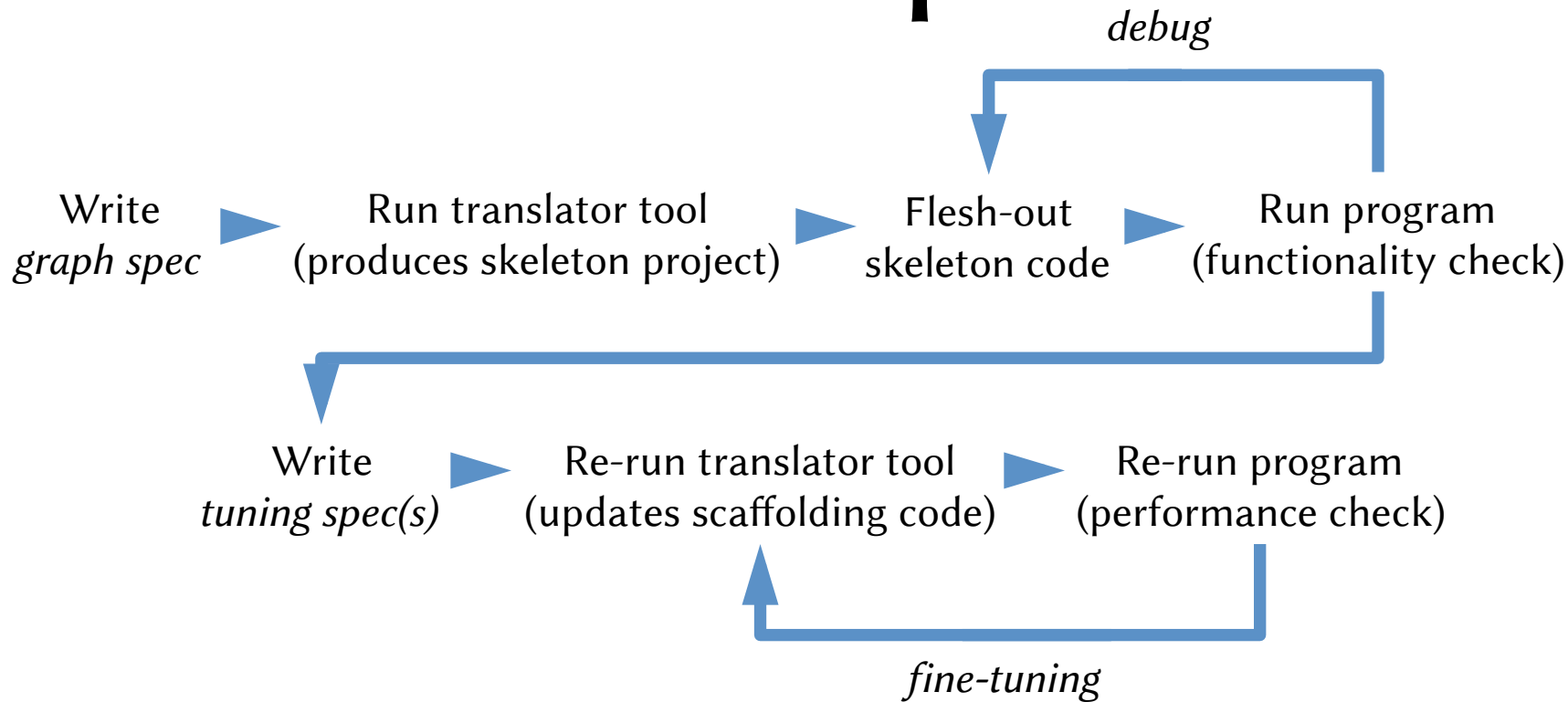
Other Considerations

- Programmability
 - Thread-local data breaks with Fibers and HCC
 - No C++, restricted C support for HCC
 - Manual transformation to Non-blocking is hard, and often performs poorly
- Debugging support
 - Valgrind breaks with Fibers
 - GDB can inspect blocked tasks with Threads

Other Considerations (cont.)

- Portability:
 - Fibers relies on platform-specific assembly
 - HCC toolchain not easily installed
- Resilience:
 - Fibers and Threads result in long-lived blocked tasks
 - Performance impact if task life exceeds MTBF

CnC-OCR Developer Workflow



CnC / OCR Concept Map

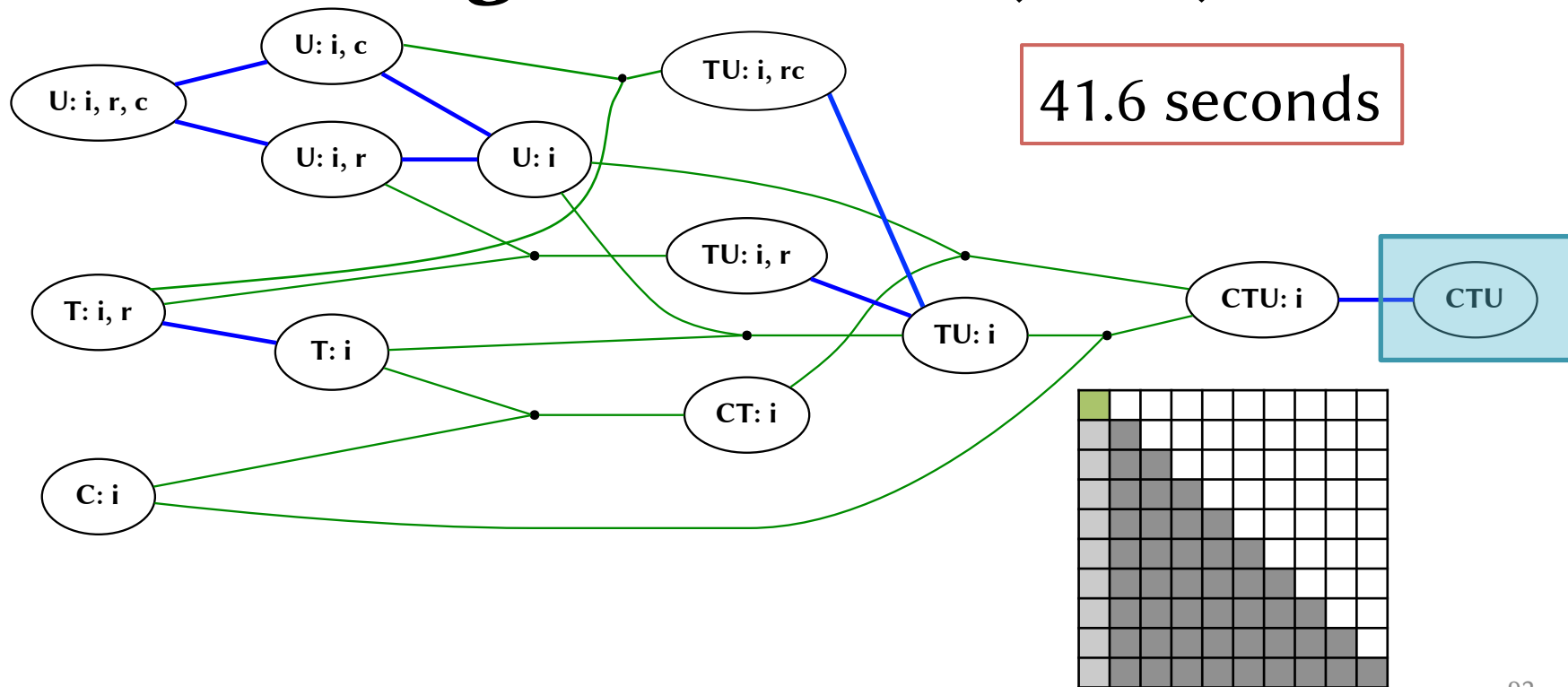
Concept	CnC construct	OCR construct
Task classes (code)	Step collection	EDT template
Task instance	Step instance	EDT
Data classes (types)	Item collection	—
Data instance	Item instance	Datablock
Unique instance identifier	Tag	GUID
Dependence registration	Item get	Event add dependence
Dependence satisfaction	Item put	Event satisfy

Extensions to the CnC Graph Specification Language

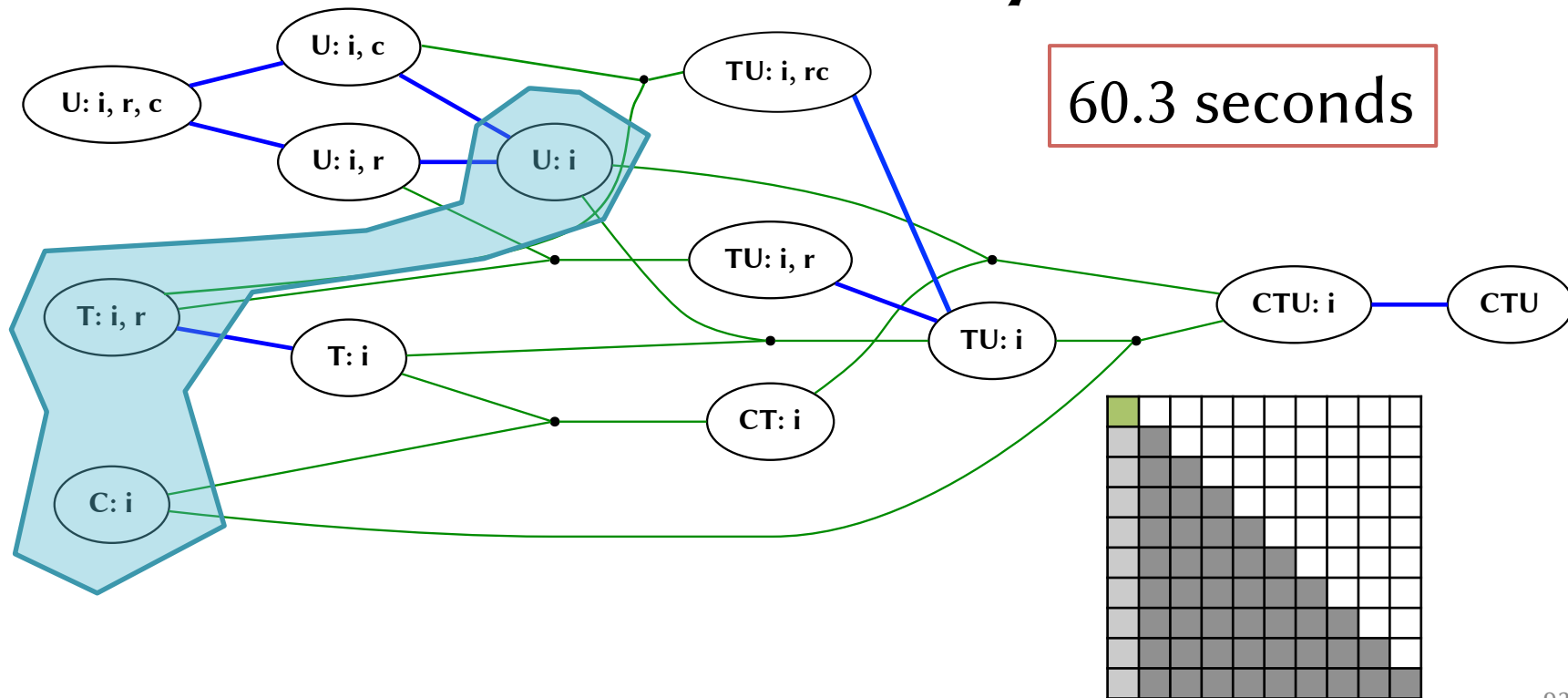
```
[ int above[#tw] : i, j ];  
[ int left[#th] : i, j ];  
[ SeqData *data : () ];  
  
( swStep: i, j )  
  <- [ data: () ],  
      [ above: i, j ] $when(i > 0),  
      [ left: i, j ] $when(j > 0)  
  -> [ below @ above: i+1, j ],  
      [ right @ left: i, j+1 ],  
      ( swStep: i+1, j )  
      $when(i+1 < #nth);
```

- ➡ Item tags (checked)
- ➡ Sized item arrays
- ➡ Singleton collections
- ➡ Conditional I/O
- ➡ Instance aliases
- ➡ Global context values
- ➡ Virtual collection views

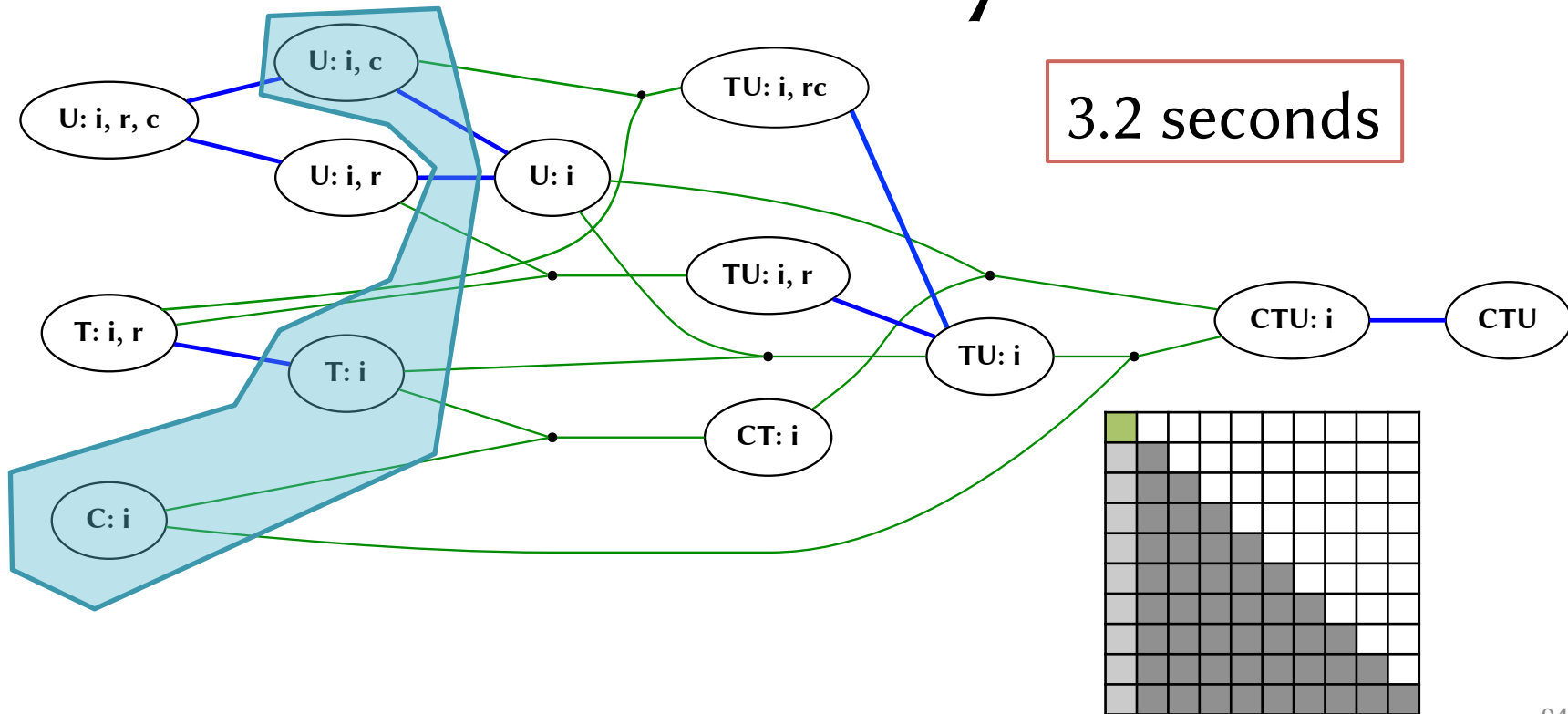
Cholesky Tuned w/CnC Hierarchy: Singleton Slice (Bad)



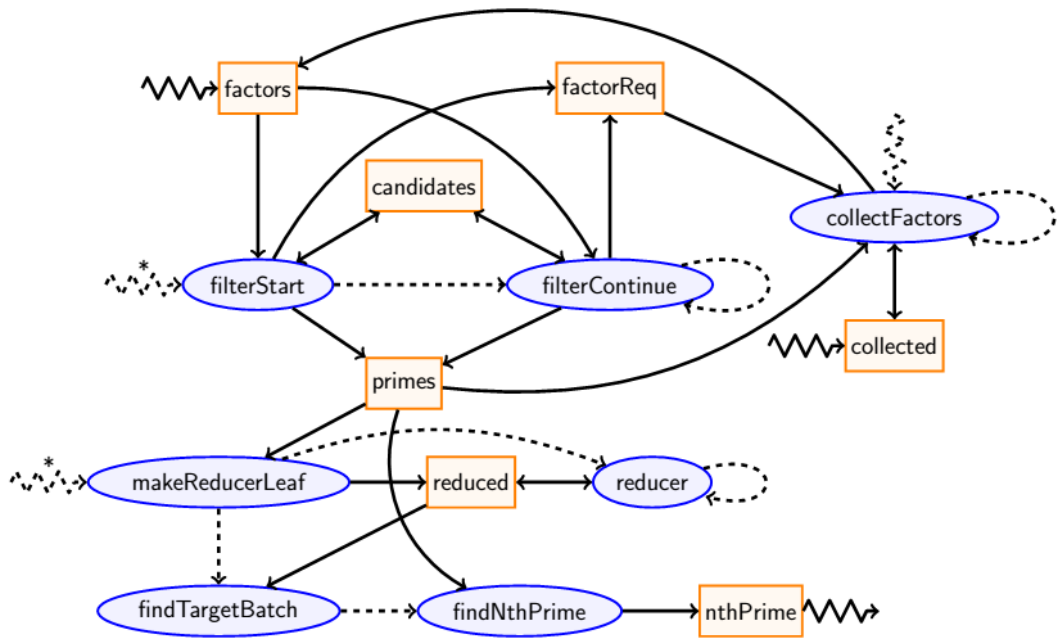
Cholesky Tuned w/CnC Hierarchy: Worst Hierarchy Slice



Cholesky Tuned w/CnC Hierarchy: Best Hierarchy Slice

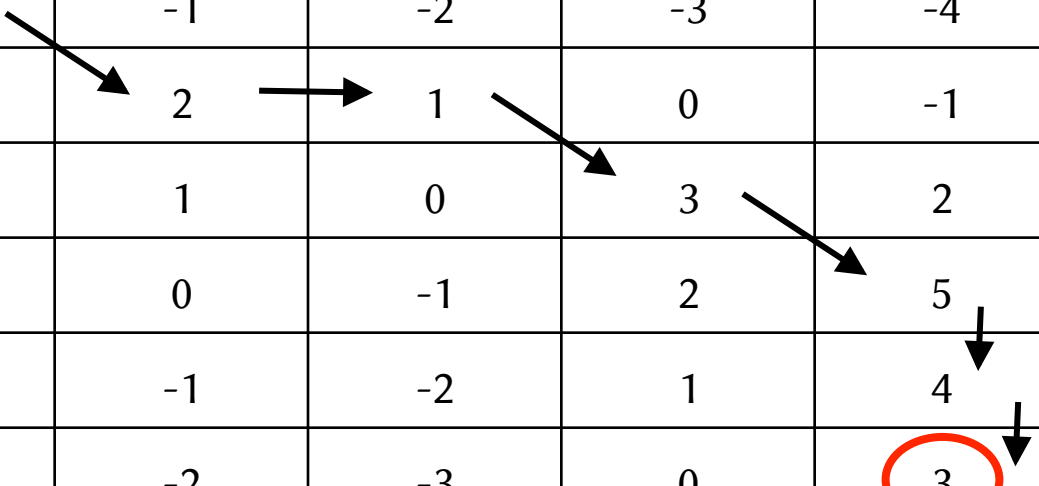


Sample CnC Graph: Nth Prime Number



CnC Example: Smith-Waterman Scoring Matrix

	--	A	G	C	A
--	0	-1	-2	-3	-4
A	-1	2	1	0	-1
C	-2	1	0	3	2
A	-3	0	-1	2	5
C	-4	-1	-2	1	4
A	-5	-2	-3	0	3



Selection of Hierarchy-based Distribution Results for Cholesky

Hierarchy Slice	Run-time*
$(CT:i) + (U:i, c)$	3.2 seconds
$(C:i) + (T:i, r) + (U:i, c)$	5.6 seconds
$(CT:i) + (U:i, r)$	9.0 seconds
$(CTU:)$	41.6 seconds
$(C:i) + (T:i, r) + (U:i)$	60.3 seconds

*Mean of 5 runs