Contech: Parallel Program Representation and High Performance Instrumentation

Brian Paul Railing
PhD Candidate in Computer Science
Georgia Institute of Technology

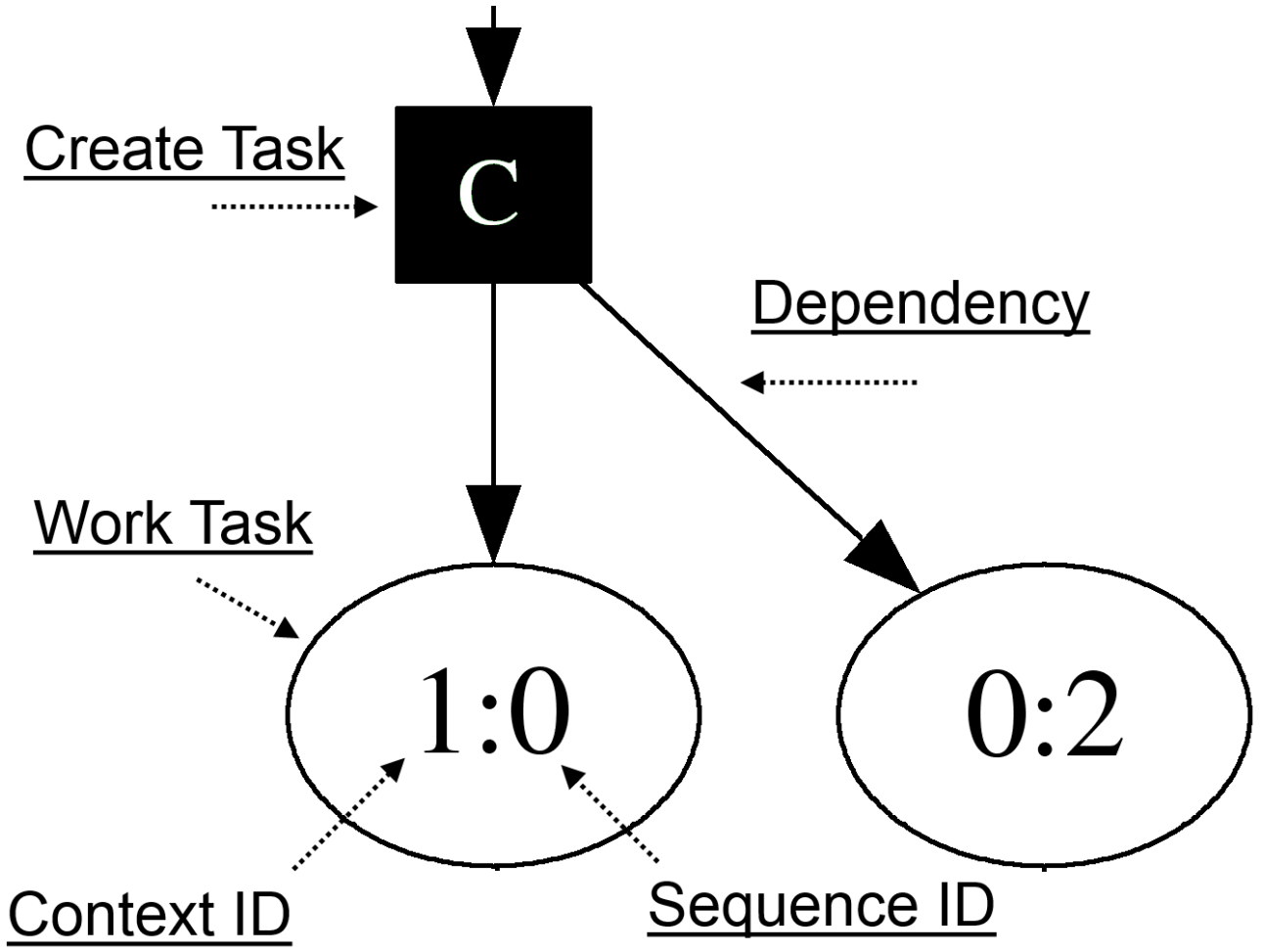
Objectives of Parallel Program Representation

- Generate the task graph with no user intervention
  - Without constraint of language, library, or structure
- A common representation needs
  - What was executed
  - What was accessed
  - In what order did threads execute
- Without recording architecture / runtime effects
  - Context switches
  - Consistency model
  - Cache Effects
  - …

Contech's Task Graph Representation

- Task Graphs are directed, acyclic graphs containing
  - Nodes partitioned based on type
  - Edges as scheduling dependencies
  - Nodes contain lists of actions and data
  - Other graph annotations such as start / end time

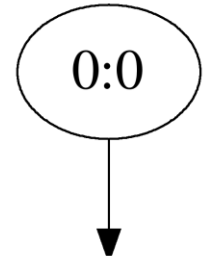
Task Graph Legend



Task Graph Example

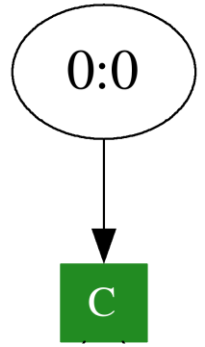
```
int fib(int n) {
  if (n < 2)
    return n;
  int a = cilk_spawn fib(n-1);
  int b = fib(n-2);
  cilk_sync;
  return a + b;
}
```

```
fib(2);
```



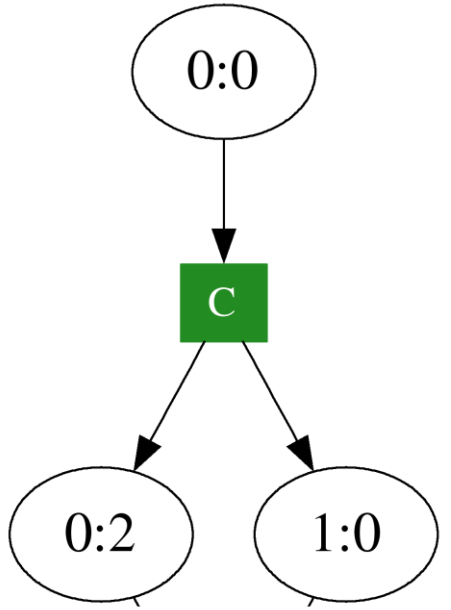
Task Graph Example

```
int fib(int n) {
   if (n < 2)
      return n;
   int a = cilk_spawn fib(n-1);
   int b = fib(n-2);
   cilk_sync;
   return a + b;
}

fib(2);
```



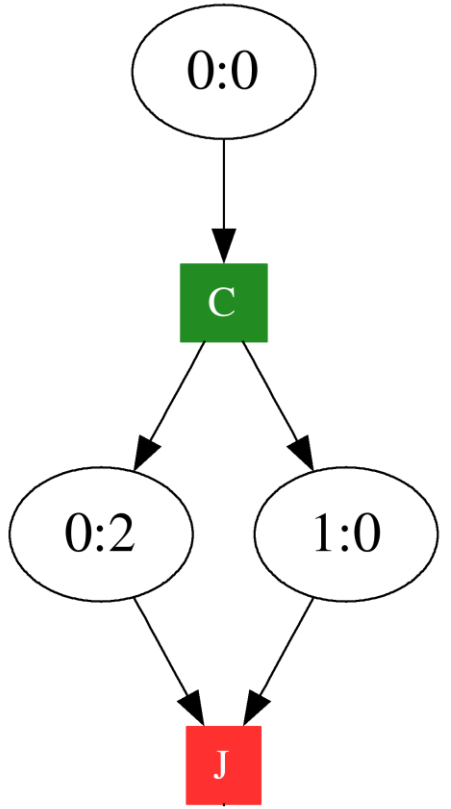
Task Graph Example

```
int fib(int n) {
   if (n < 2)
      return n;
   int a = cilk_spawn fib(n-1);
   int b = fib(n-2);
   cilk_sync;
   return a + b;
}

fib(2);
```



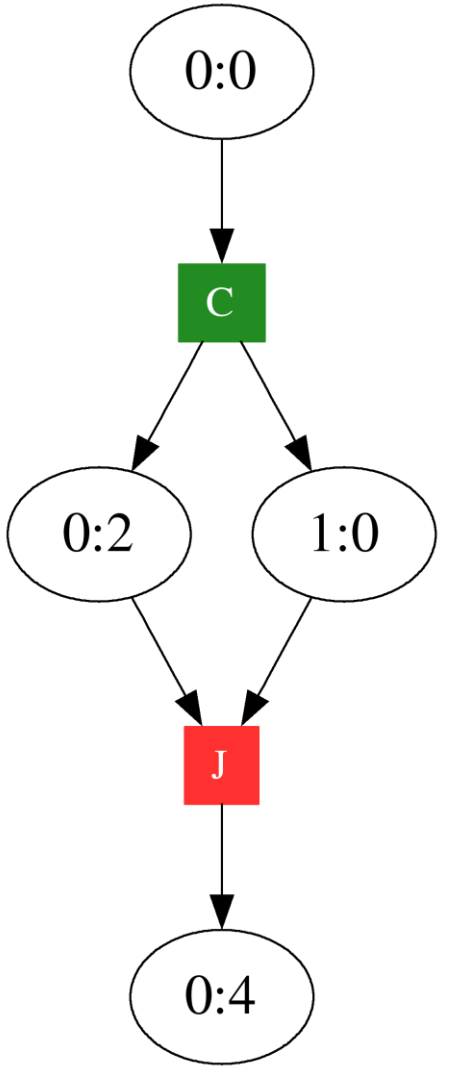
Task Graph Example

```
int fib(int n) {
   if (n < 2)
      return n;
   int a = cilk_spawn fib(n-1);
   int b = fib(n-2);
   cilk_sync;
   return a + b;
}

fib(2);
```



Task Graph Example

```
int fib(int n) {
   if (n < 2)
      return n;
   int a = cilk_spawn fib(n-1);
   int b = fib(n-2);
   cilk_sync;
   return a + b;
}

fib(2);
```



Parallel Program Diversity

- Language Diversity
  - **C, C++, Fortran**, Java, Go, Rust, X10, …
- Runtime Diversity
  - **Pthreads, OpenMP, MPI, Cilk**, Galois, Legion, CnC, …
- Pattern Diversity
  - **Regular, pipelines, graphs**, Map-reduce, Gather-scatter, …
- Architecture Diversity
  - **32- / 64-bit x86, ARM**, MIPS, Power, …

Outline

- Representing Parallel Programs
- **State of the art collection**
- Parallel Program Analysis
- Conclusion

Collecting the Representation

- Compiler-based instrumentation using LLVM
  - Primarily, Clang frontend to generate LLVM IR
- Link against runtime library
  - Implementation of instrumentation routines
  - Background support for writing the trace
- Convert the trace into a task graph

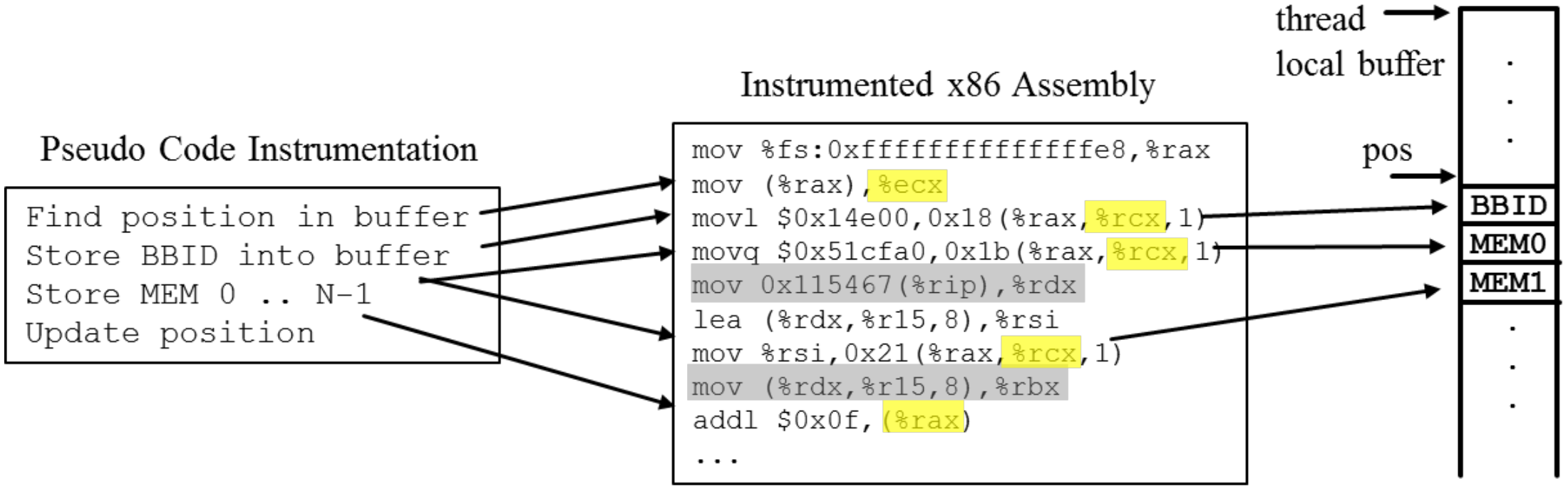
Instrumenting a Program

- Contech LLVM pass instruments IR of interest
  - Every basic block
  - Loads / Stores
  - Calls to functions of interest
    - Memory management (malloc, free, new, delete, memcpy, etc)
    - Pthreads (pthread_create, pthread_mutex_lock, etc)
    - OpenMP, …

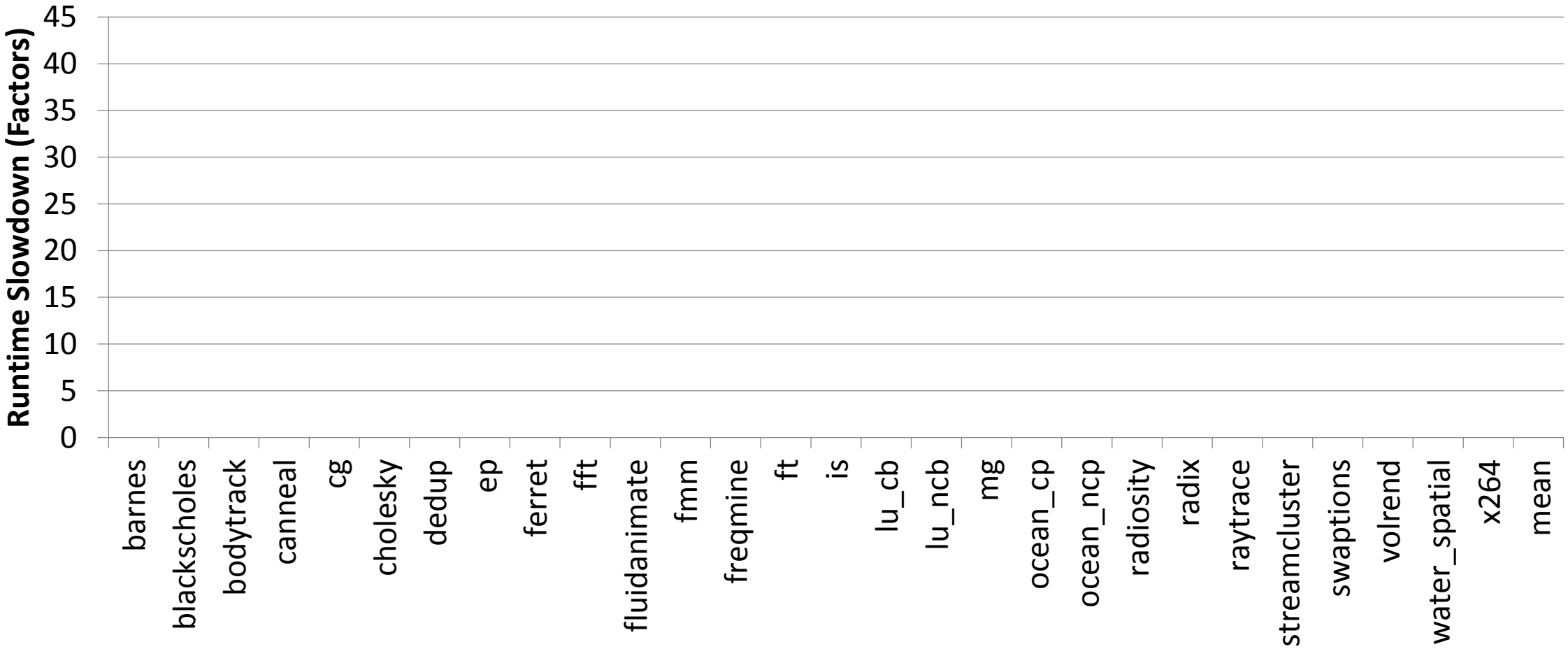
Instrumenting IR of Interest

- Call Contech instrumentation routines (~40 in number)
  - For example `__ctStoreBasicBlock(i32 474, i32 %bufPos3, i8* %bufPos2)`
  - Instrumentation written in C
    - Architecture independent (32- / 64-bit x86, 32-bit ARM)
  - Instrumentation routines are co-designed
- Use Clang's link time optimizer (LTO)
  - Inline these calls into short assembly sequences

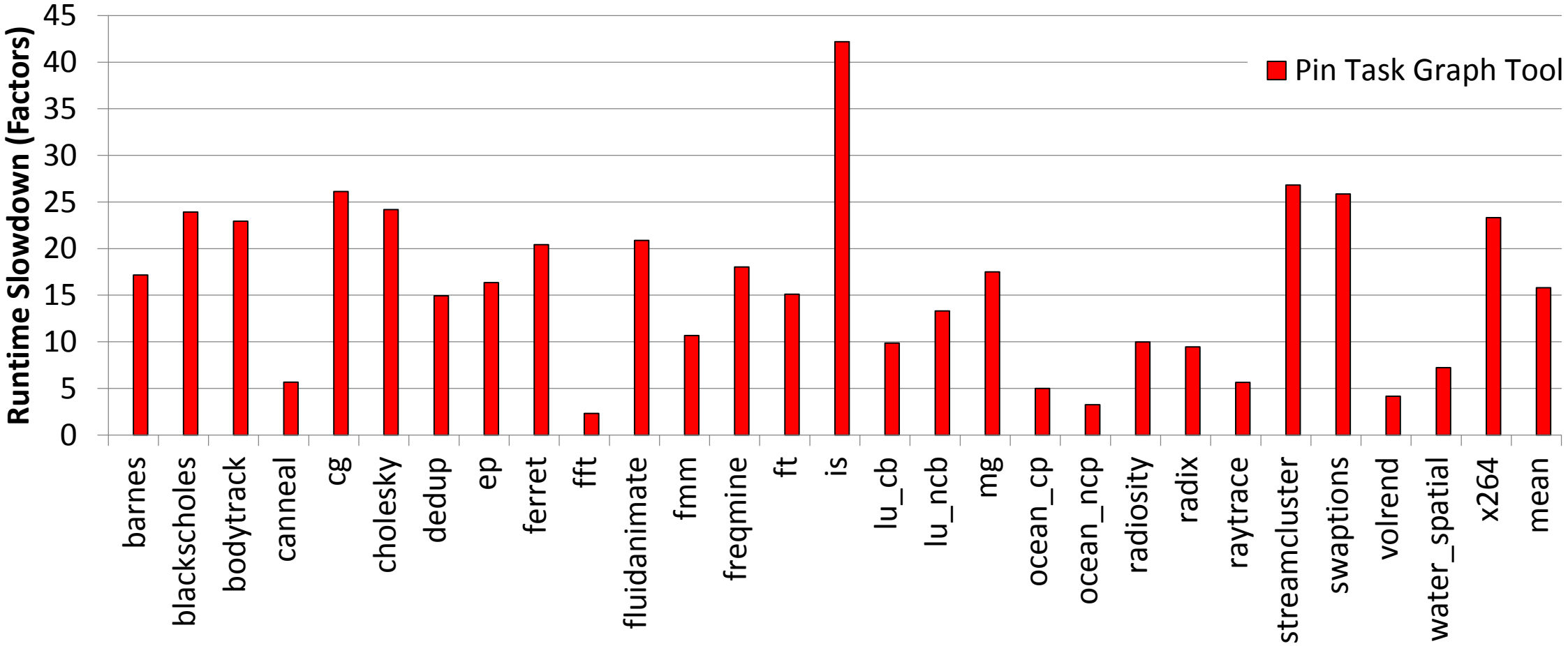
Instrumentation Design



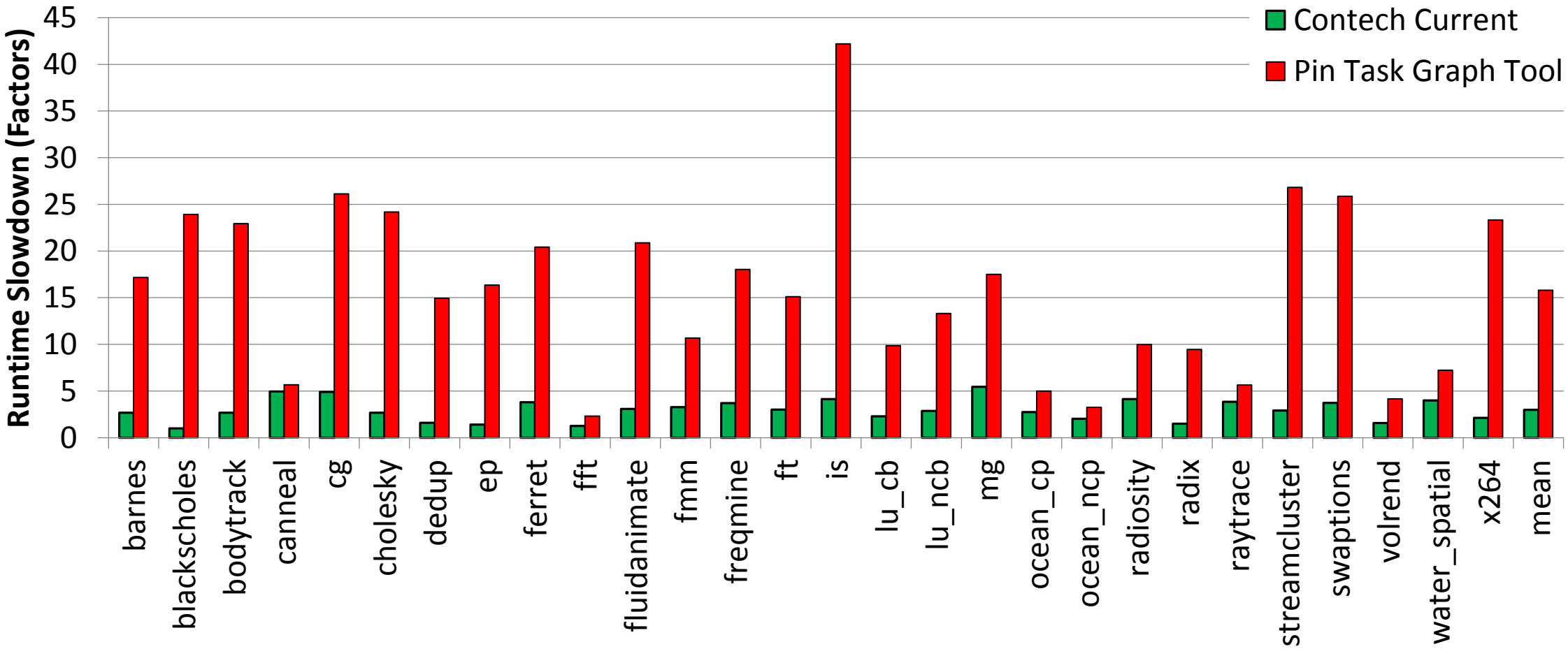
Slowdown (CPU Overhead)



Slowdown (CPU Overhead)



Slowdown (CPU Overhead)



Contech Instrumentation Summary

- 3x average instrumentation overhead
- Tested Support for:
  - C, C++, Fortran
  - x86, ARM
  - PThreads, OpenMP, MPI, Cilk

Outline

- Representing Parallel Programs
- State of the art collection
- **Parallel Program Analysis**
- Conclusion

Rich Analysis Support

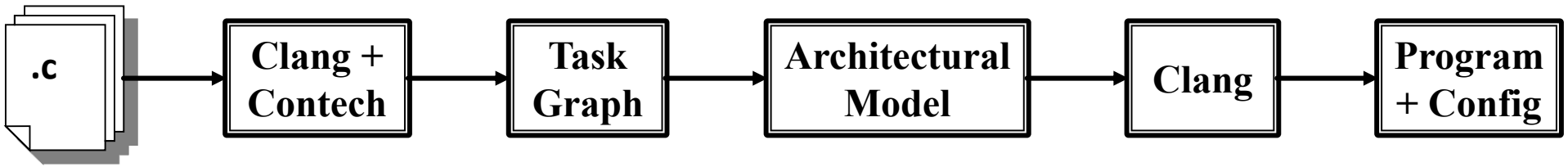
- Compiler-based framework to generate task graphs
- Example analysis of Task Graphs
  - Data Race Detection
  - Cache modeling
  - Communication Profiling
  - Lock contention

Propose a Reconfigurable Architecture

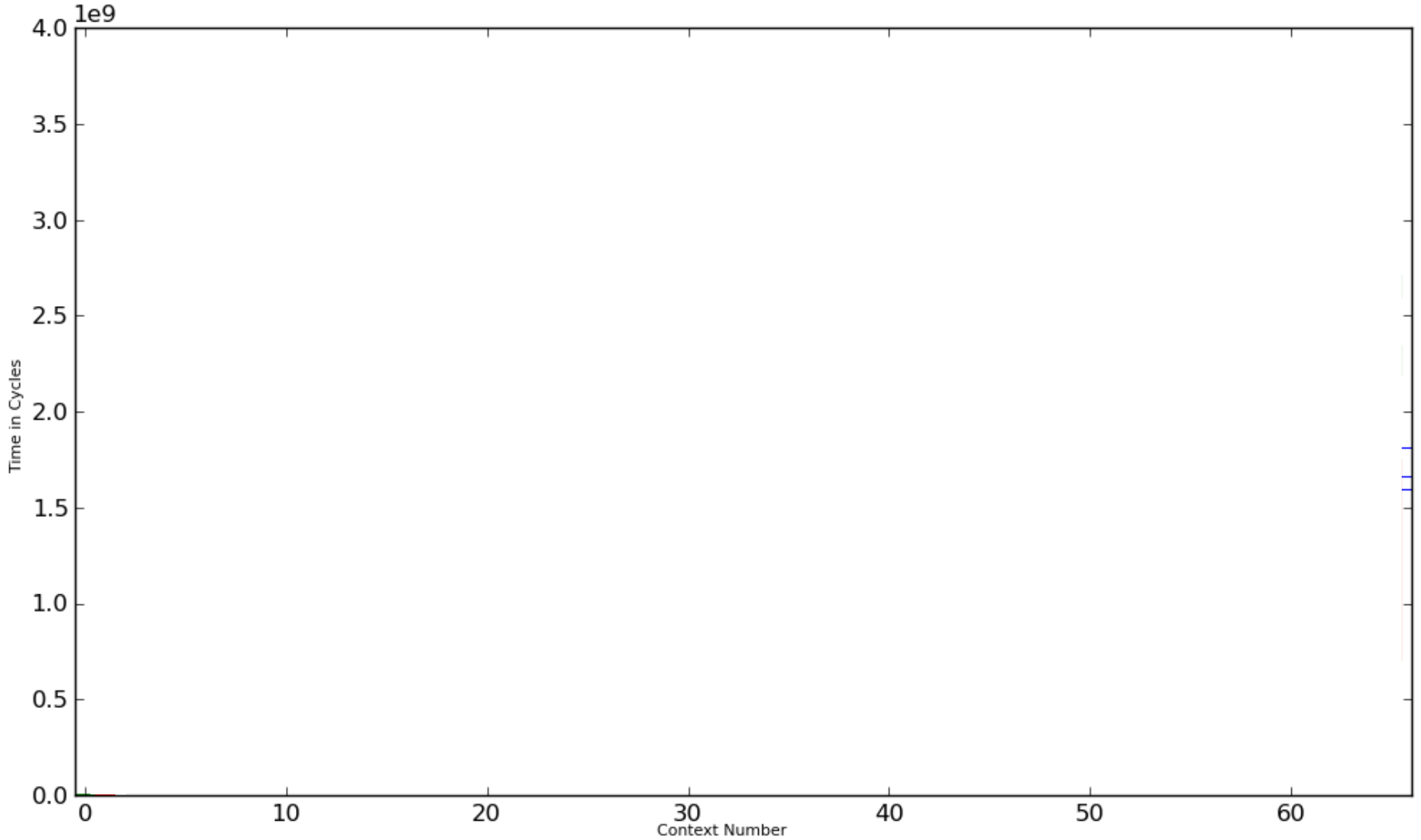
- Take a standard, superscalar processor
  - Add additional functional units that can be enabled / disabled
  - Add the support for software to specify unit requirements
- Asymmetric multiprocessing system
  - Processors are provisioned over the thermal budget
  - Each processor can be reconfigured separately

Optimization Workflow

- Use Contech to generate a task graph for analysis
- Prior architectural model that determines resource constraints
- Recompile program with adaptation hints
  - Each hint is associated with an edge in the CFG



Results Example - ferret



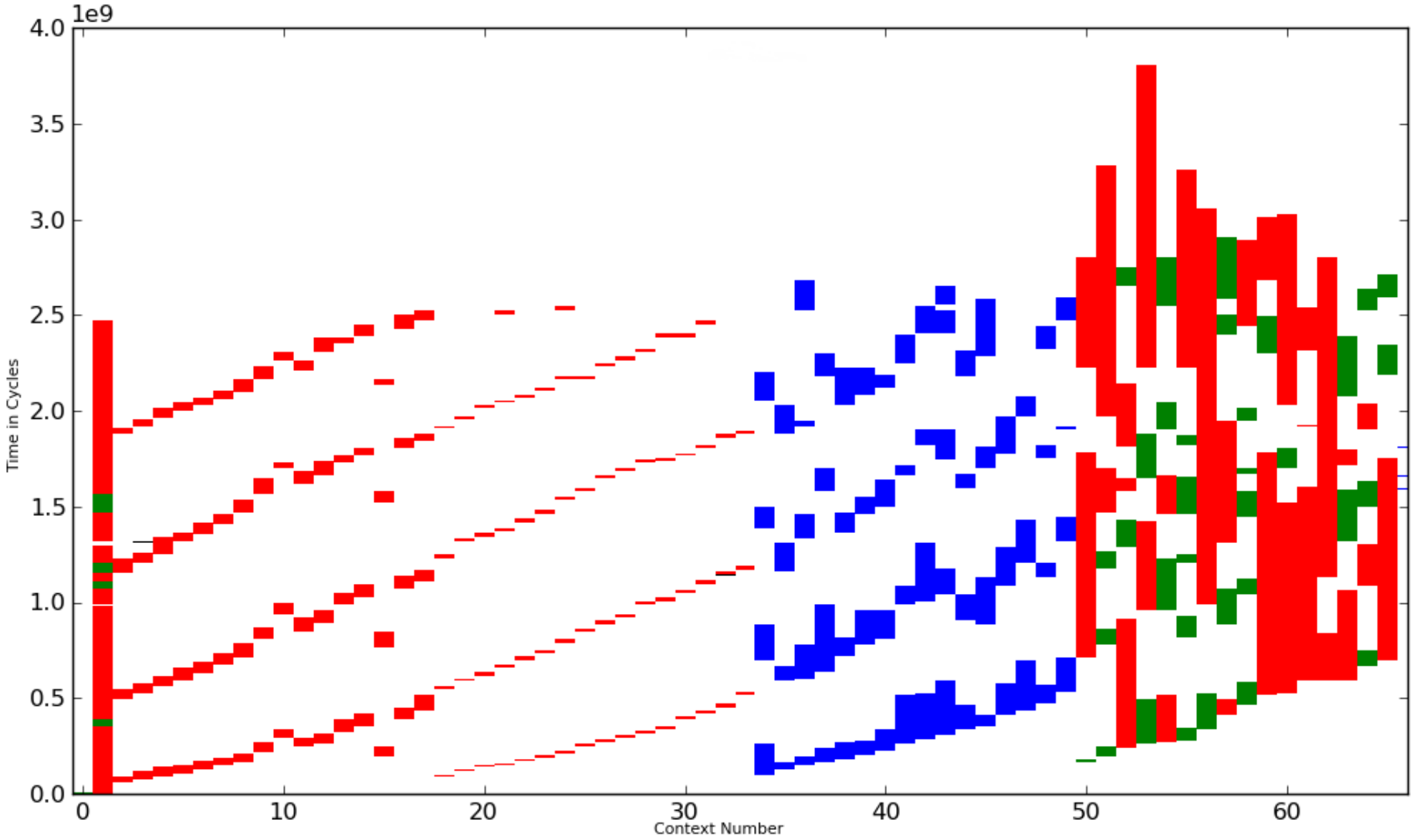
LEGEND

Floating Point

Integer

Memory

Results Example - ferret

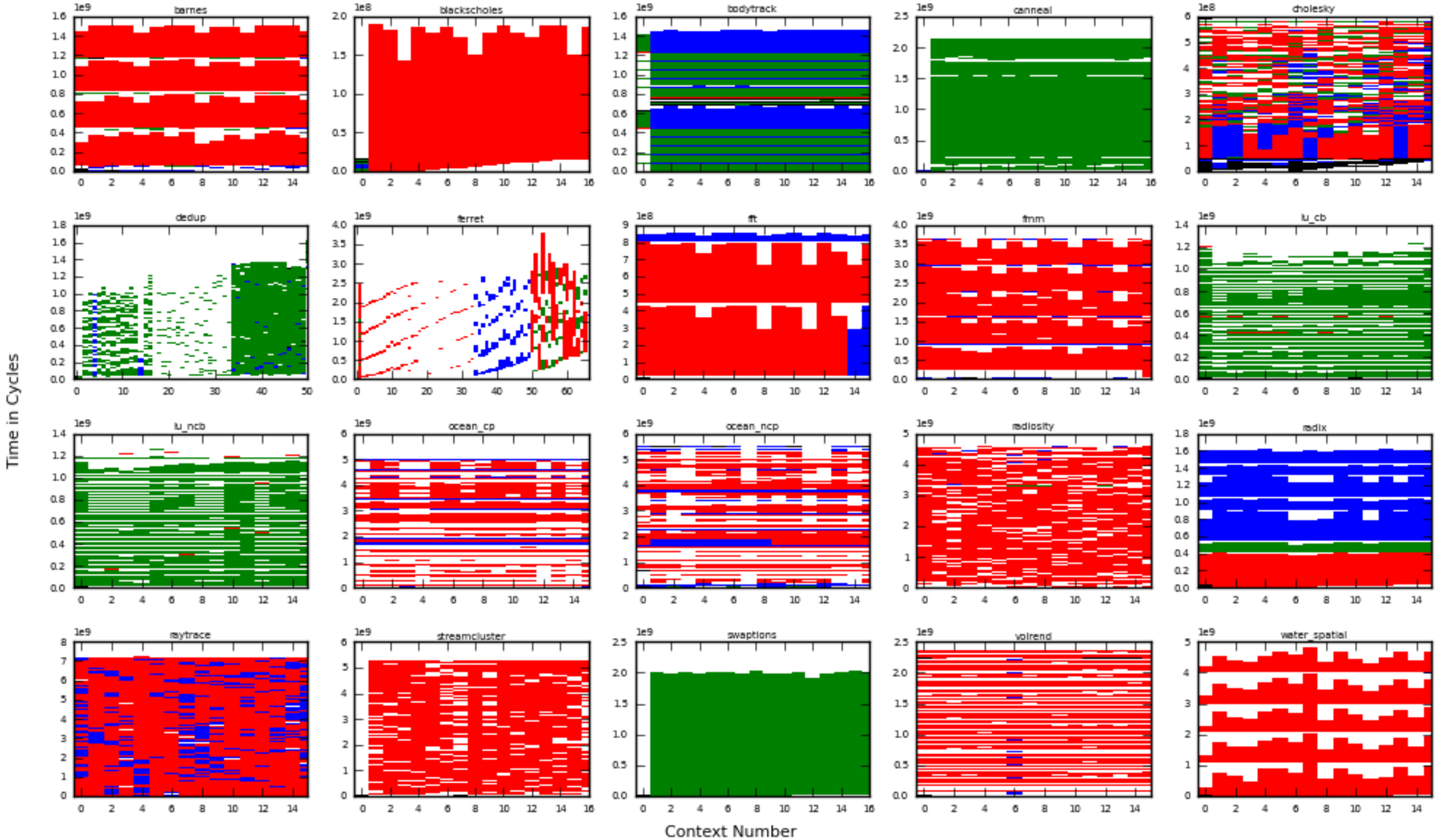


LEGEND

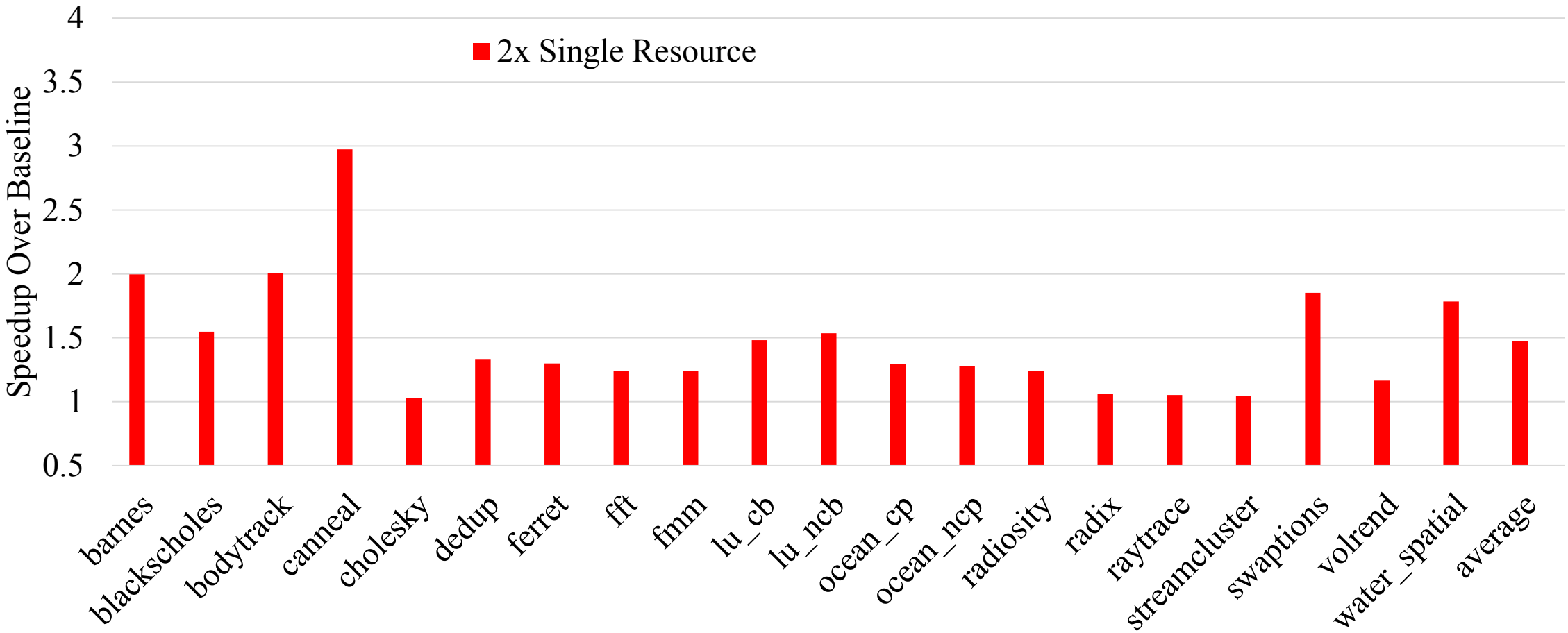
Floating Point

Integer

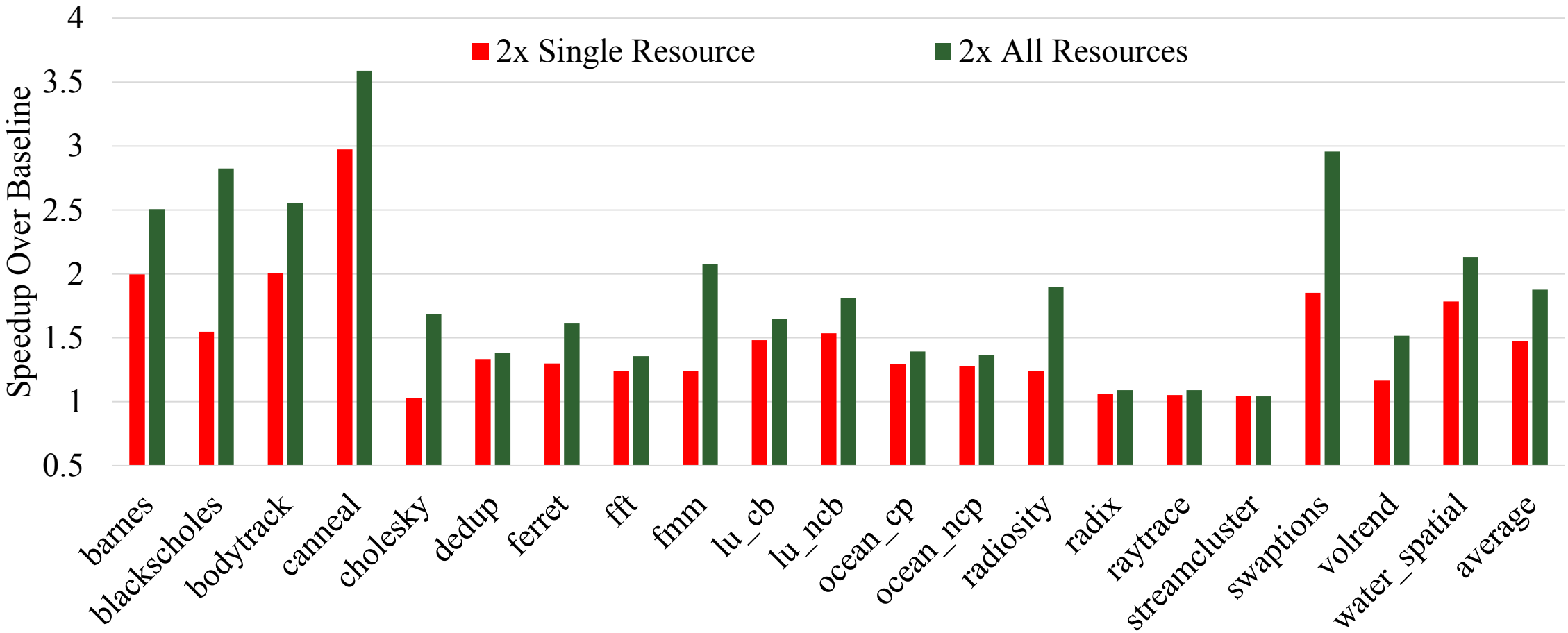
Memory


barnes
blackscholes
bodytrack
canneal
cholesky
dedup
ferret
fft
fmm
lu_cb
lu_ncb
ocean_cp
ocean_ncp
radiosity
radix
raytrace
streamcluster
swaptions
volrend
water_spatial
Time in Cycles
Context Number

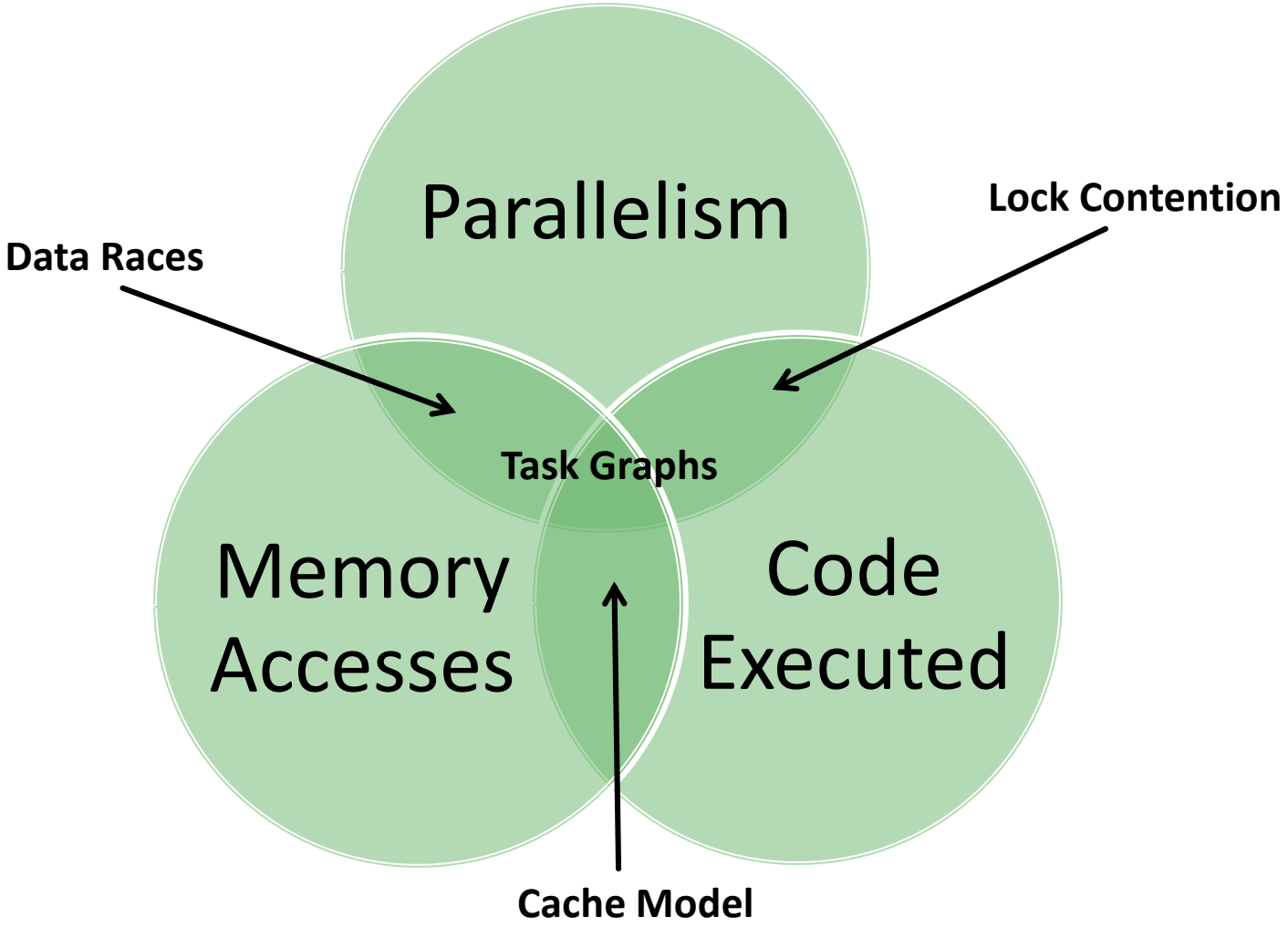
Performance Gains



Performance Gains



Analysis Features



Outline

- Representing Parallel Programs
- State of the art collection
- Parallel Program Analysis
- **Conclusion**

Acknowledgements

- Contech Collaborators
  - Thomas Conte (advisor)
  - Eric Hein
  - Sriseshan Srikanth
- Model Collaborators
  - Victoria Cabezas (ETH Zurich)

- Sriseshan Srikanth, **Brian P. Railing**, and Thomas M. Conte, "Dynamically reconfiguring multi-core architectures using task graph based analysis," (abstract only) To appear in Proceedings of the 24th International Conference on Parallel Architectures and Compilation, PACT '15, (New York, NY, USA), ACM, 2015.
- **Brian P. Railing**, Eric R. Hein, and Thomas M. Conte. 2015. *Contech: Efficiently Generating Dynamic Task Graphs for Arbitrary Parallel Programs.* ACM Trans. Archit. Code Optim. 12, 2, Article 25 (July 2015), 24 pages.
- **Brian P. Railing**. 2015. Using Active-Learning Techniques in Mixed Undergraduate / Graduate Courses (abstract only). In *Proceedings of the 46th ACM technical symposium on Computer science education* (SIGCSE '15). ACM, New York, NY, USA.
- Rishiraj A. Bheda, Jesse G. Beu, **Brian P. Railing**, and Thomas M. Conte. 2012. Extrapolation Pitfalls When Evaluating Limited Endurance Memory. In Proceedings of the 2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '12). IEEE Computer Society, Washington, DC, USA, 261-268.
- Changhee Jung, Silvius Rus, **Brian P. Railing**, Nathan Clark, and Santosh Pande. 2011. Brainy: effective selection of data structures. In Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation (PLDI '11). ACM, New York, NY, USA, 86-97.
- Jason A. Poovey, **Brian P. Railing**, and Thomas M. Conte. 2011. Parallel pattern detection for architectural improvements. In *Proceedings of the 3rd USENIX conference on Hot topic in parallelism* (HotPar'11). USENIX Association, Berkeley, CA, USA, 12-12.

Conclusions

- Program diversity captured in the Task Graph representation
- Minimal program perturbation from instrumentation
- Representation supports program analysis

Conclusions

- Program diversity captured in the Task Graph representation
- Minimal program perturbation from instrumentation
- Representation supports program analysis

Thank you