github.com/arcana-lab/heartbeatcompiler

Compiling Loop-Based Nested Parallelism for Irregular Workloads

Yian Su, Mike Rainey, Nick Wanninger, Nadharm Dhiantravan, Jasper Liang, Umut A. Acar, Peter Dinda, Simone Campanoni

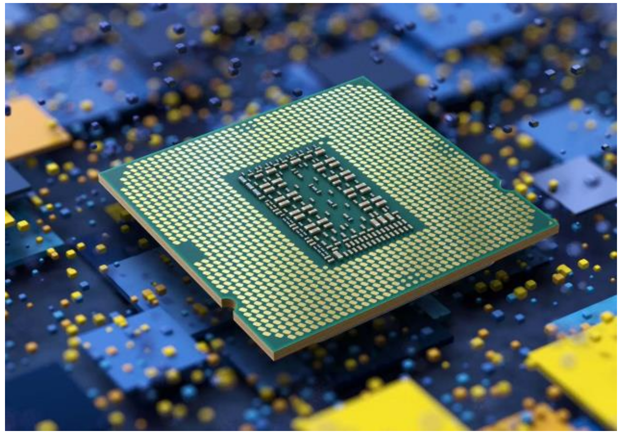
ARCANA Rethinks Compiler Abstractions for New Applications

Northwestern University

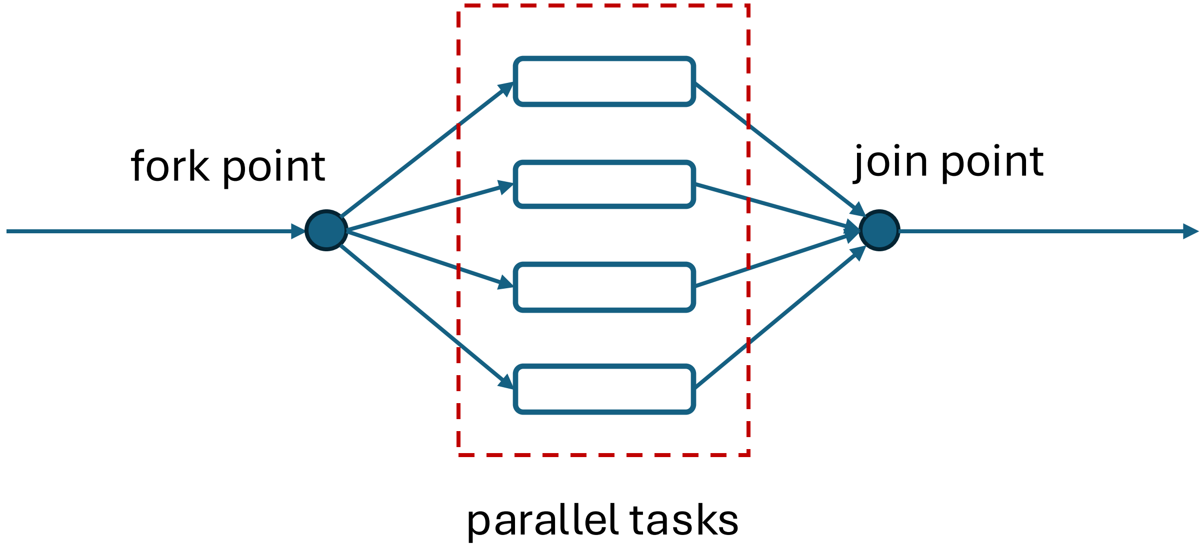
Carnegie Mellon University

Parallelism is Mainstream

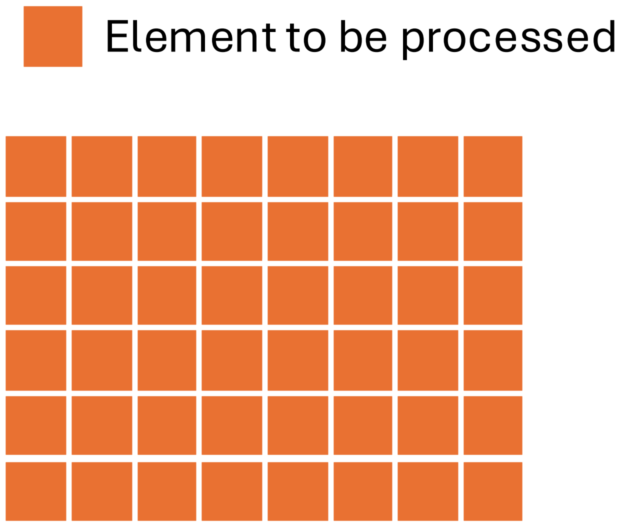
OpenMP®



Fork-Join Parallelism

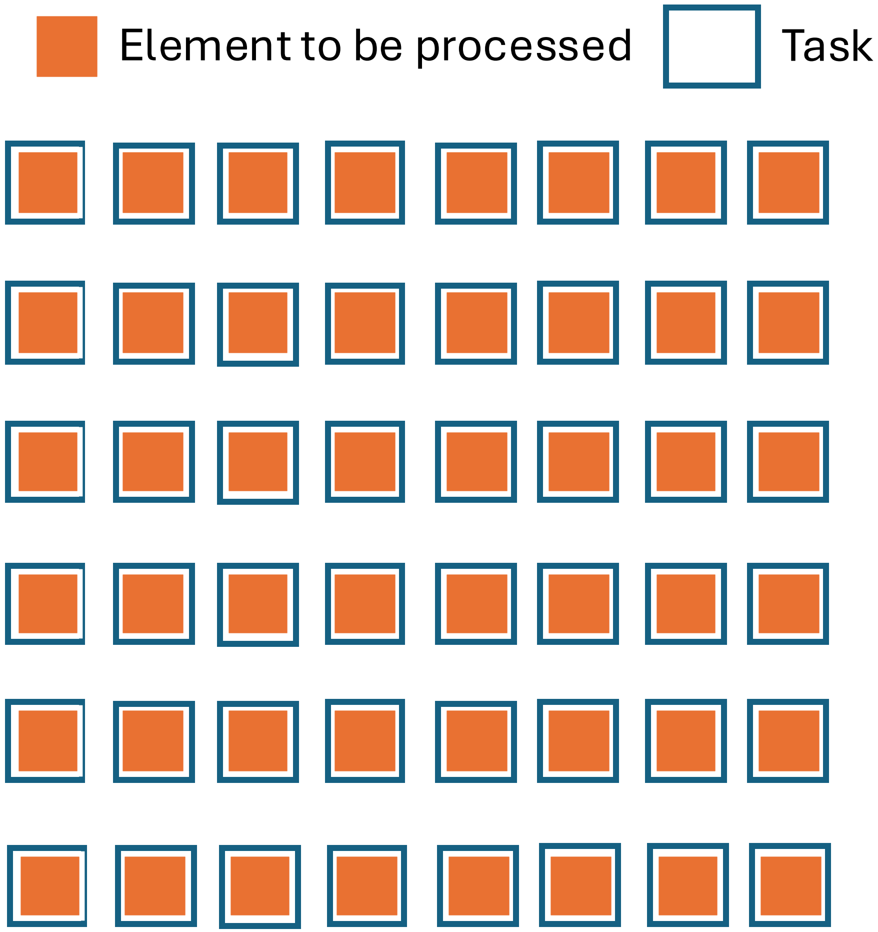


```
#pragma omp parallel for
for (int i = 0; i < M.num_rows(); i++) {
  initialWork();
  #pragma omp parallel for
  for (int j = 0; j < M.num_nonzeros(i); j++) {
    processElement(M, i, j);
  }
  writeResult();
}
```

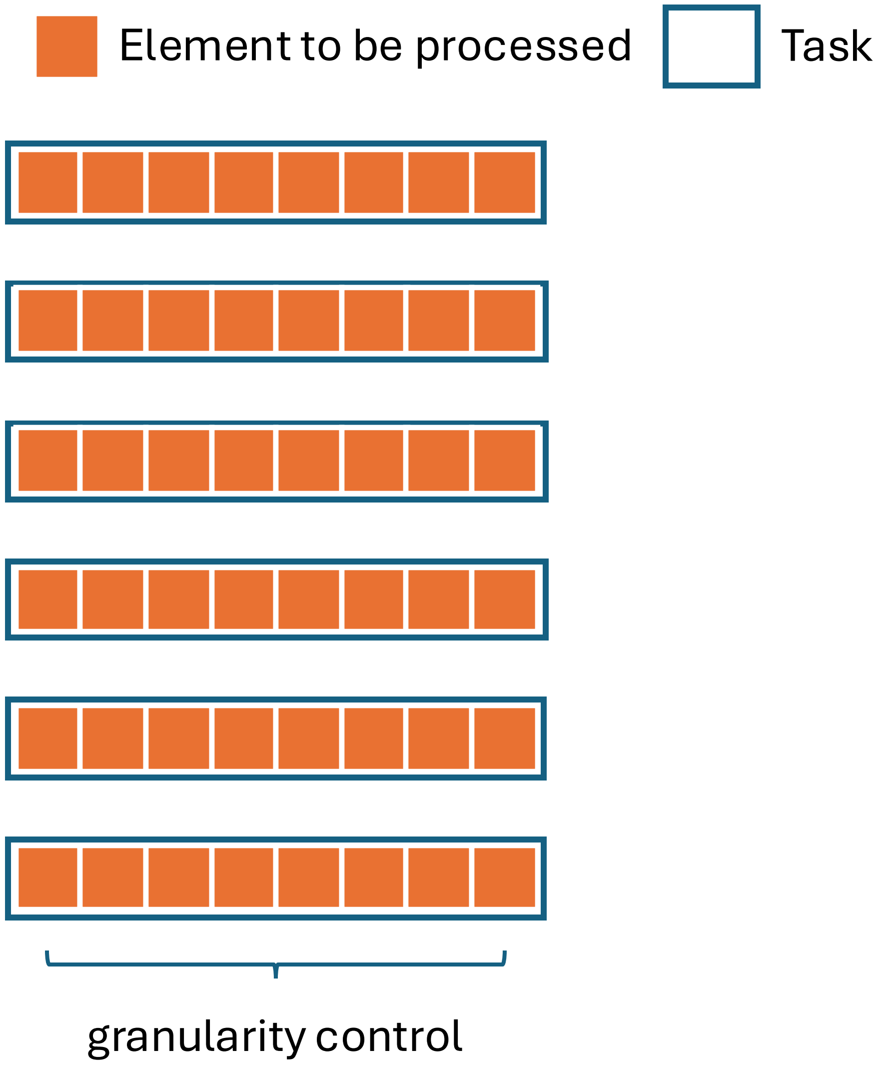


Matrix M

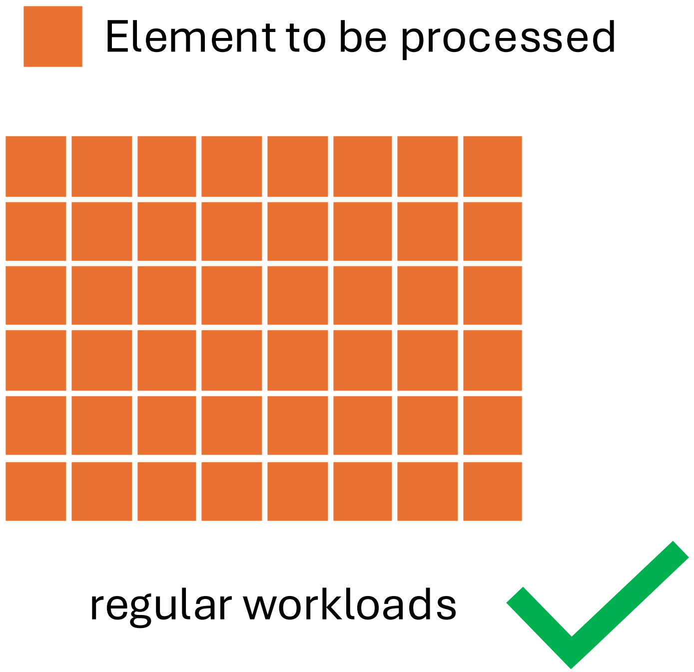
```
#pragma omp parallel for
for (int i = 0; i < M.num_rows(); i++) {
  initialWork();
  #pragma omp parallel for
  for (int j = 0; j < M.num_nonzeros(i); j++) {
    processElement(M, i, j);
  }
  writeResult();
}
```



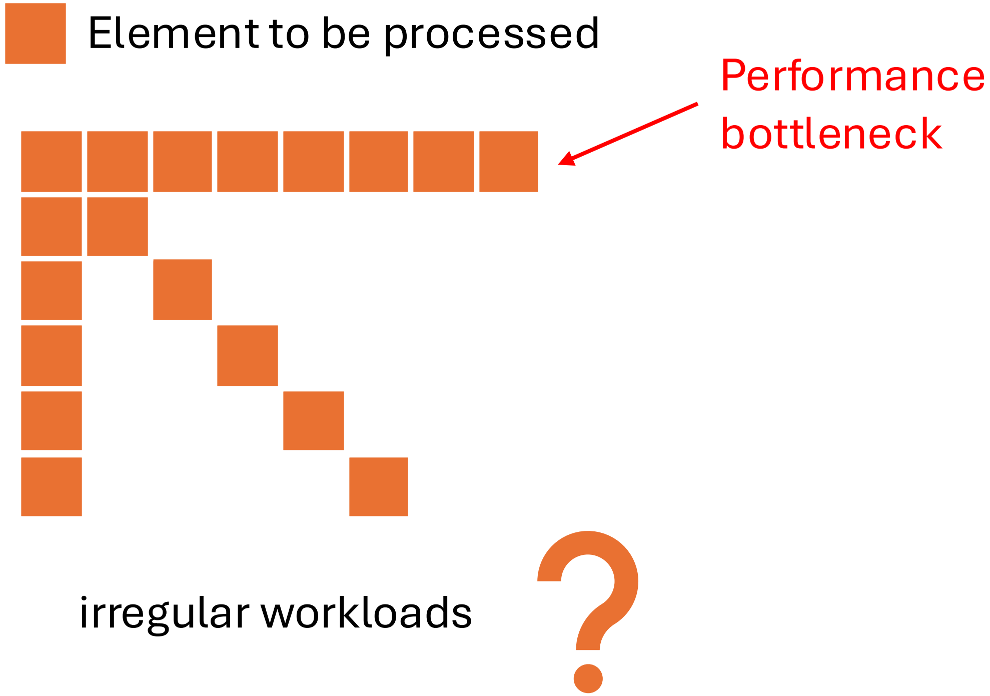
```
#pragma omp parallel for
for (int i = 0; i < M.num_rows(); i++) {
  initialWork();
  #pragma omp parallel for
  for (int j = 0; j < M.num_nonzeros(i); j++) {
    processElement(M, i, j);
  }
  writeResult();
}
```



```
#pragma omp parallel for
for (int i = 0; i < M.num_rows(); i++) {
  initialWork();
  #pragma omp parallel for
  for (int j = 0; j < M.num_nonzeros(i); j++) {
    processElement(M, i, j);
  }
  writeResult();
}
```

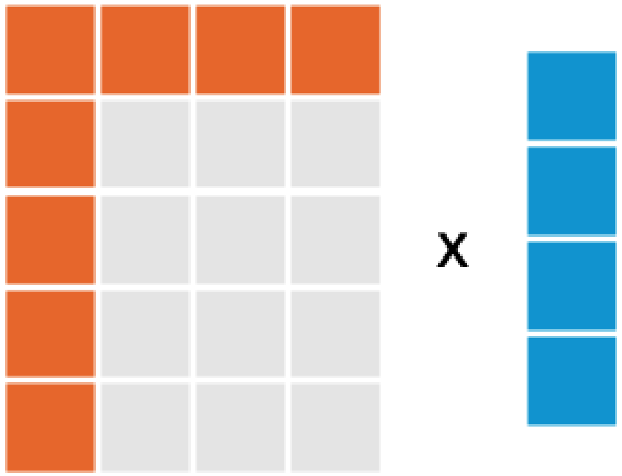


```
#pragma omp parallel for
for (int i = 0; i < M.num_rows(); i++) {
  initialWork();
  #pragma omp parallel for
  for (int j = 0; j < M.num_nonzeros(i); j++) {
    processElement(M, i, j);
  }
  writeResult();
}
```

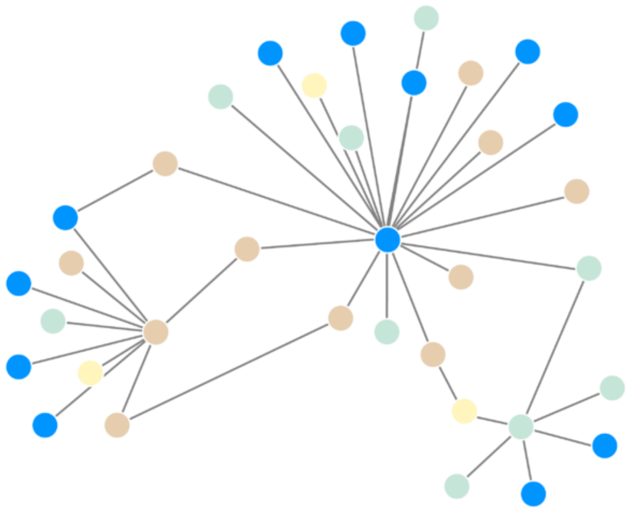


Irregular Workloads

Sparse Tensor Algebra



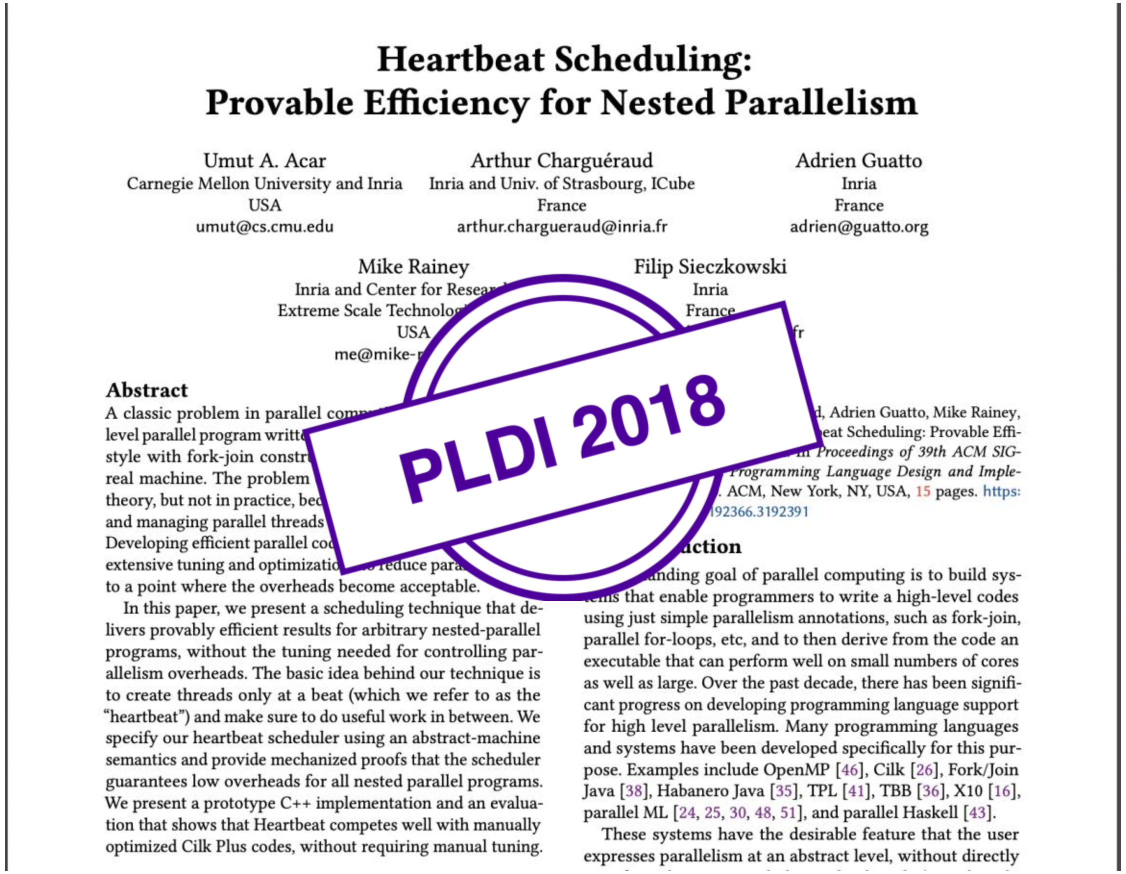
Graph Analytics



Existing Solutions

- OpenMP static scheduler
- OpenMP dynamic scheduler
- OpenMP guided scheduler

Heartbeat Scheduling



Heartbeat Scheduling: Provable Efficiency for Nested Parallelism

Umut A. Acar
Carnegie Mellon University and Inria
USA
umut@cs.cmu.edu

Arthur Charguéraud
Inria and Univ. of Strasbourg, ICube
France
arthur.chargueraud@inria.fr

Adrien Guatto
Inria
France
adrien@guatto.org

Mike Rainey
Inria and Center for Resea
Extreme Scale Technolog
USA
me@mike-

Filip Sieczkowski
Inria
France
fr

Abstract

A classic problem in parallel com
level parallel program writt
style with fork-join constr
real machine. The problem
theory, but not in practice, bec
and managing parallel threads
Developing efficient parallel co
extensive tuning and optimizatio to reduce para
to a point where the overheads become acceptable.

In this paper, we present a scheduling technique that delivers provably efficient results for arbitrary nested-parallel programs, without the tuning needed for controlling parallelism overheads. The basic idea behind our technique is to create threads only at a beat (which we refer to as the "heartbeat") and make sure to do useful work in between. We specify our heartbeat scheduler using an abstract-machine semantics and provide mechanized proofs that the scheduler guarantees low overheads for all nested parallel programs. We present a prototype C++ implementation and an evaluation that shows that Heartbeat competes well with manually optimized Cilk Plus codes, without requiring manual tuning.

d, Adrien Guatto, Mike Rainey,
eat Scheduling: Provable Effi-
Proceedings of 39th ACM SIG-
Programming Language Design and Imple-
. ACM, New York, NY, USA, 15 pages. https:
192366.3192391

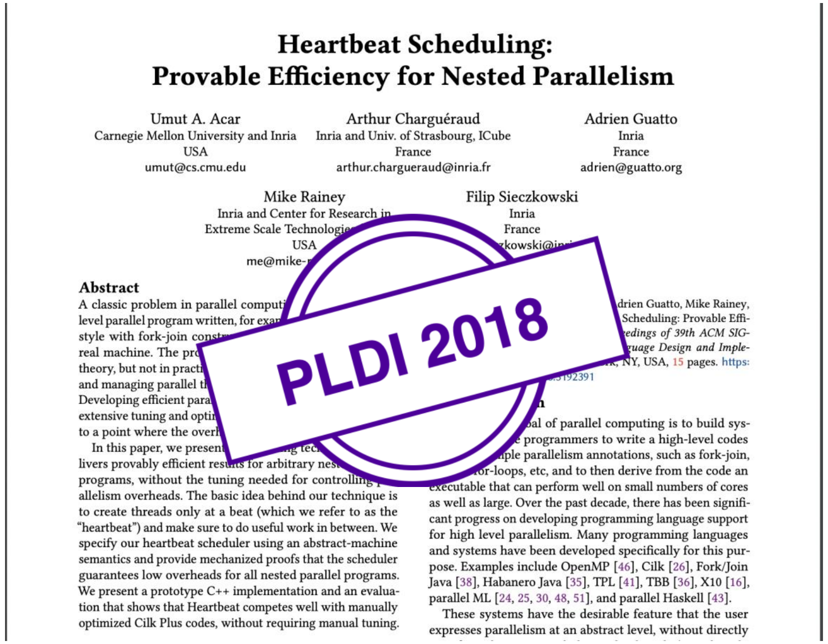
...ction

...anding goal of parallel computing is to build systems that enable programmers to write a high-level codes using just simple parallelism annotations, such as fork-join, parallel for-loops, etc, and to then derive from the code an executable that can perform well on small numbers of cores as well as large. Over the past decade, there has been significant progress on developing programming language support for high level parallelism. Many programming languages and systems have been developed specifically for this purpose. Examples include OpenMP [46], Cilk [26], Fork/Join Java [38], Habanero Java [35], TPL [41], TBB [36], X10 [16], parallel ML [24, 25, 30, 48, 51], and parallel Haskell [43].

These systems have the desirable feature that the user expresses parallelism at an abstract level, without directly

automatic granularity control ✓

performance guarantees ✓


Heartbeat Scheduling: Provable Efficiency for Nested Parallelism

Umut A. Acar
Carnegie Mellon University and Inria
USA
umut@cs.cmu.edu

Arthur Charguéraud
Inria and Univ. of Strasbourg, ICube
France
arthur.chargueraud@inria.fr

Adrien Guatto
Inria
France
adrien@guatto.org

Mike Rainey
Inria and Center for Research in
Extreme Scale Technologie
USA
me@mike-

Filip Sieczkowski
Inria
France
zkowski@inri

Abstract

A classic problem in parallel computi level parallel program written, for exa style with fork-join const real machine. The pr theory, but not in practi and managing parallel t Developing efficient para extensive tuning and opti to a point where the overh

In this paper, we present livers provably efficient results for arbitrary nest programs, without the tuning needed for controlling allelism overheads. The basic idea behind our technique is to create threads only at a beat (which we refer to as the "heartbeat") and make sure to do useful work in between. We specify our heartbeat scheduler using an abstract-machine semantics and provide mechanized proofs that the scheduler guarantees low overheads for all nested parallel programs. We present a prototype C++ implementation and an evaluation that shows that Heartbeat competes well with manually optimized Cilk Plus codes, without requiring manual tuning.

drien Guatto, Mike Rainey, Scheduling: Provable Effi- edings of 39th ACM SIG- uage Design and Imple- k, NY, USA, 15 pages. https: 3192391

al of parallel computing is to build sys- programmers to write a high-level codes ple parallelism annotations, such as fork-join, for-loops, etc, and to then derive from the code an executable that can perform well on small numbers of cores as well as large. Over the past decade, there has been significant progress on developing programming language support for high level parallelism. Many programming languages and systems have been developed specifically for this purpose. Examples include OpenMP [46], Cilk [26], Fork/Join Java [38], Habanero Java [35], TPL [41], TBB [36], X10 [16], parallel ML [24, 25, 30, 48, 51], and parallel Haskell [43].

These systems have the desirable feature that the user expresses parallelism at an abstract level, without directly

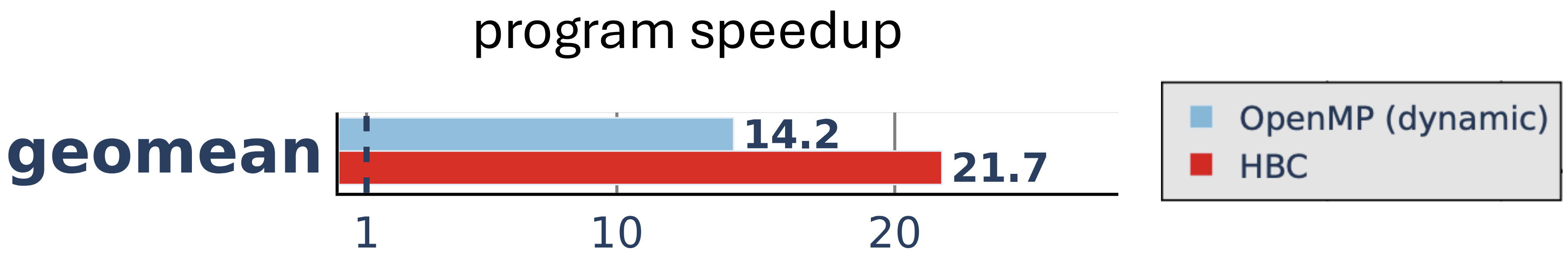
automatic granularity control

performance guarantees

No Automation

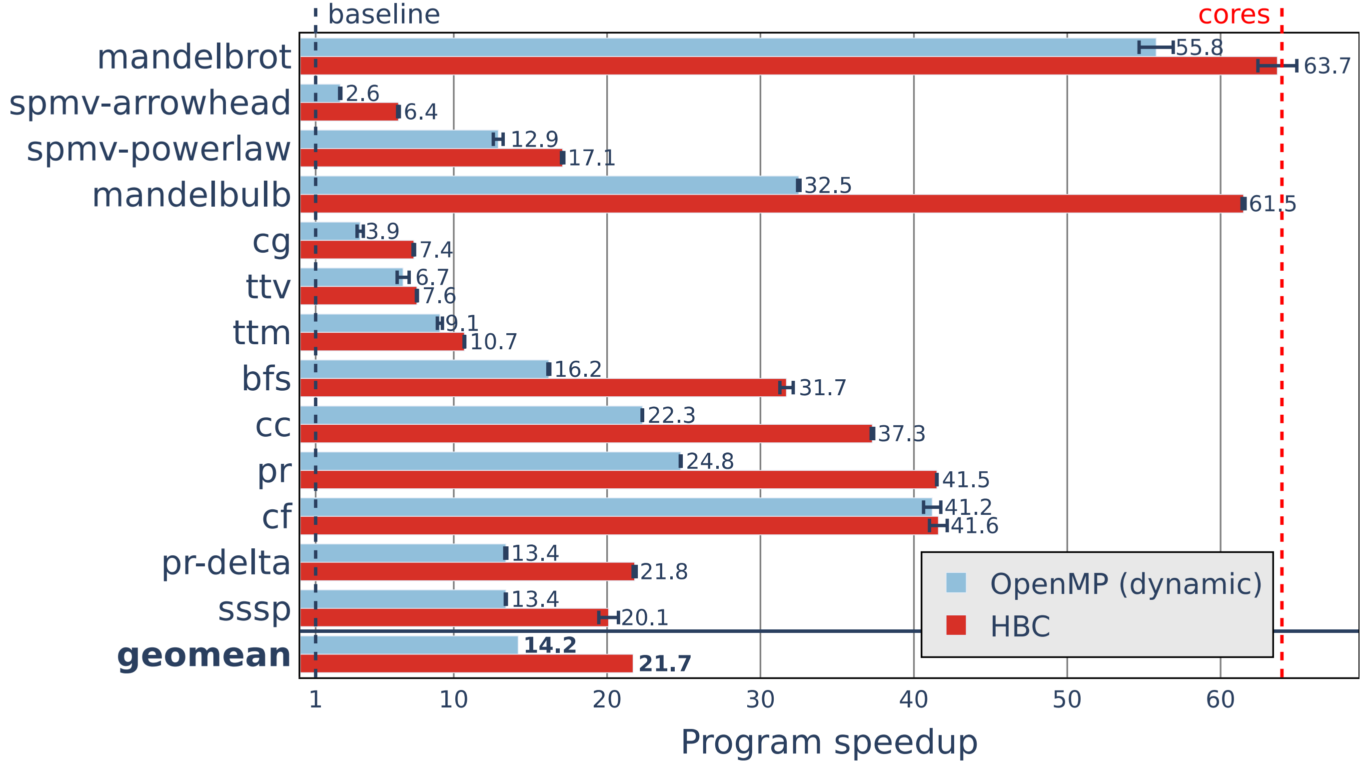
Our Work

Heartbeat Compiler (HBC)



Higher is better

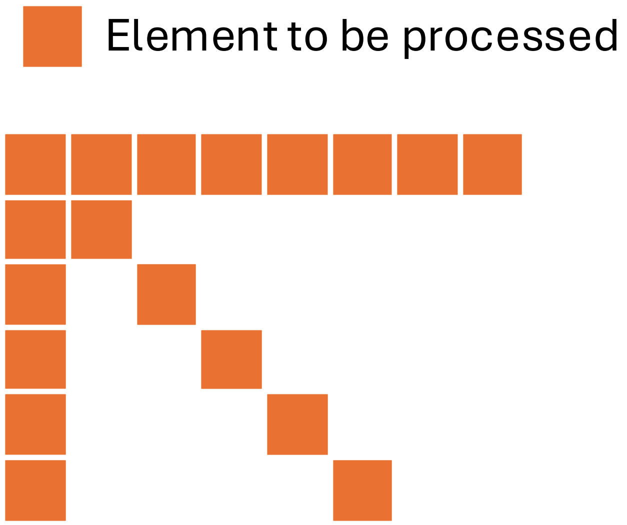
HBC for Irregular Workloads



HBC outperforms the state-of-the-art granularity control solutions for irregular workloads!

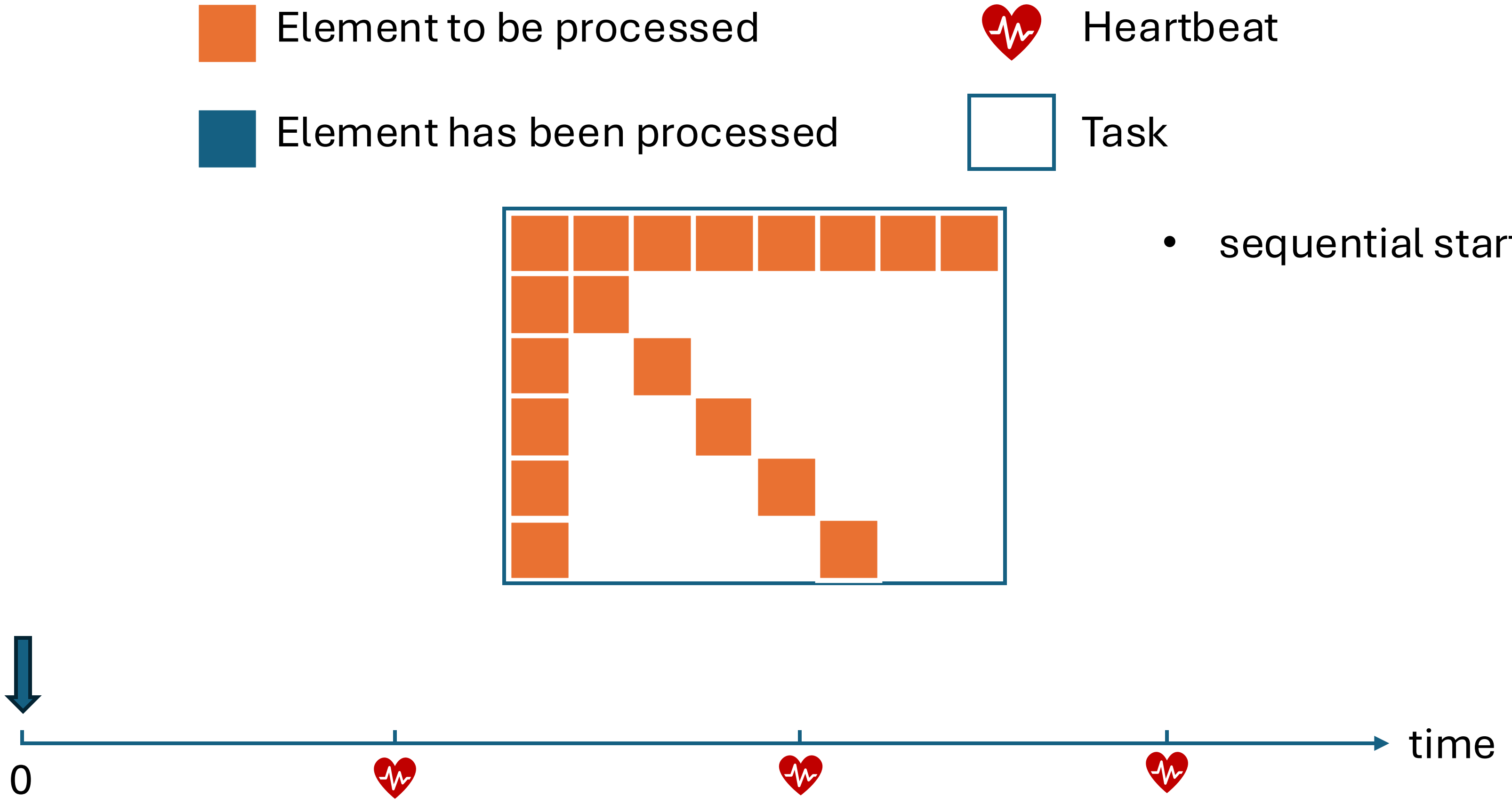
Background: Heartbeat Scheduling

```
#pragma omp parallel for
for (int i = 0; i < M.num_rows(); i++) {
  initialWork();
  #pragma omp parallel for
  for (int j = 0; j < M.num_nonzeros(i); j++) {
    processElement(M, i, j);
  }
  writeResult();
}
```



[1] Umut A. Acar, Arthur Charguéraud, Adrien Guatto, Mike Rainey, and Filip Sieczkowski. Heartbeat scheduling: provable efficiency for nested parallelism. PLDI 2018

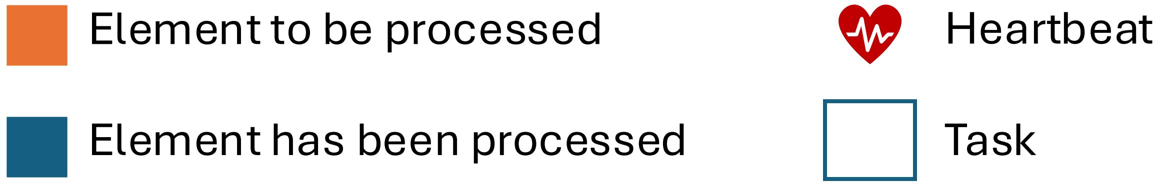
Background: Heartbeat Scheduling

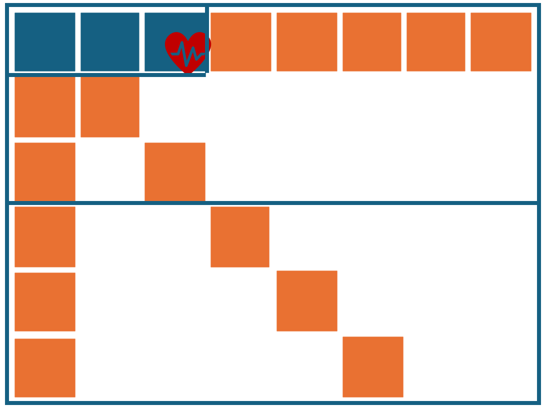


- sequential start

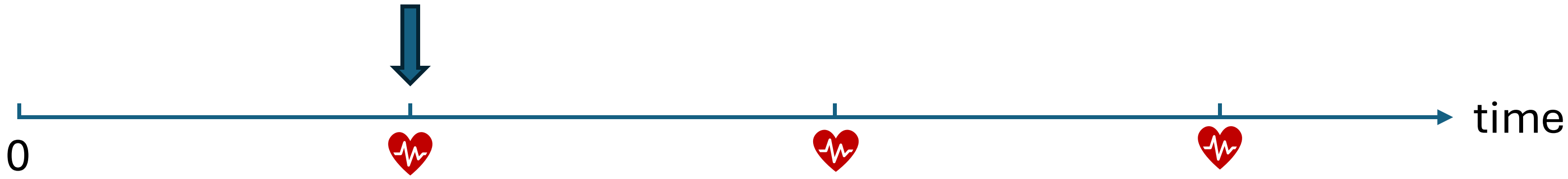
[1] Umut A. Acar, Arthur Charguéraud, Adrien Guatto, Mike Rainey, and Filip Sieczkowski. Heartbeat scheduling: provable efficiency for nested parallelism. PLDI 2018

Background: Heartbeat Scheduling



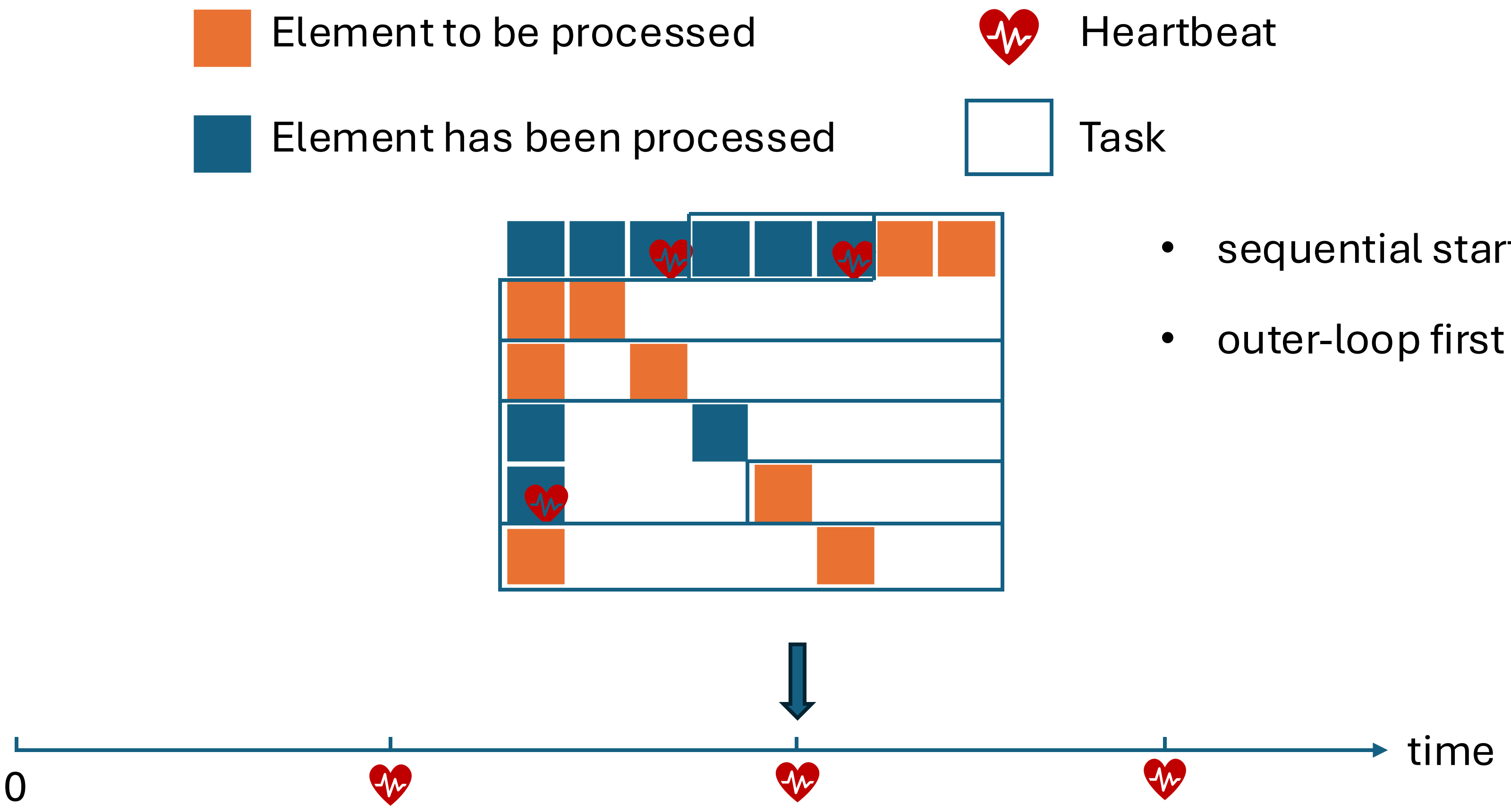


- sequential start
- outer-loop first



[1] Umut A. Acar, Arthur Charguéraud, Adrien Guatto, Mike Rainey, and Filip Sieczkowski. Heartbeat scheduling: provable efficiency for nested parallelism. PLDI 2018

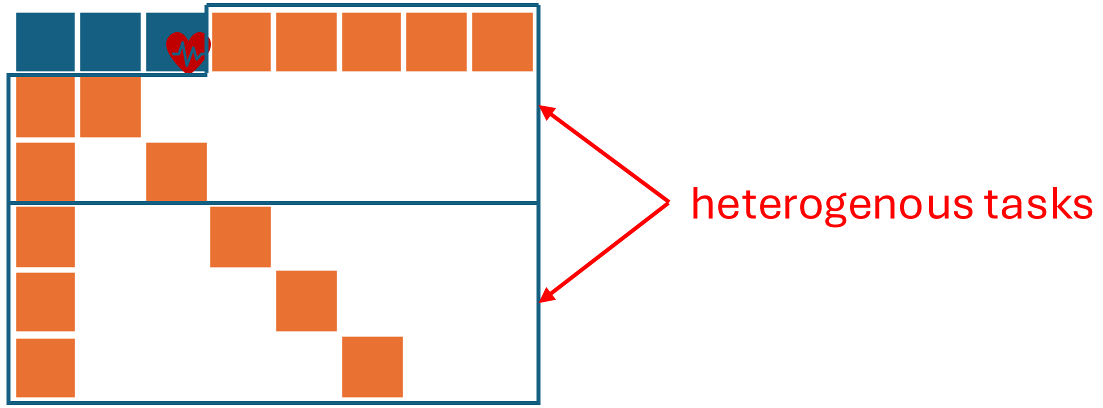
Background: Heartbeat Scheduling



- sequential start
- outer-loop first

[1] Umut A. Acar, Arthur Charguéraud, Adrien Guatto, Mike Rainey, and Filip Sieczkowski. Heartbeat scheduling: provable efficiency for nested parallelism. PLDI 2018

```
#pragma omp parallel for
for (int i = 0; i < M.num_rows(); i++) {
  initialWork();
  #pragma omp parallel for
  for (int j = 0; j < M.num_nonzeros(i); j++) {
    processElement(M, i, j);
  }
  writeResult();
}
```



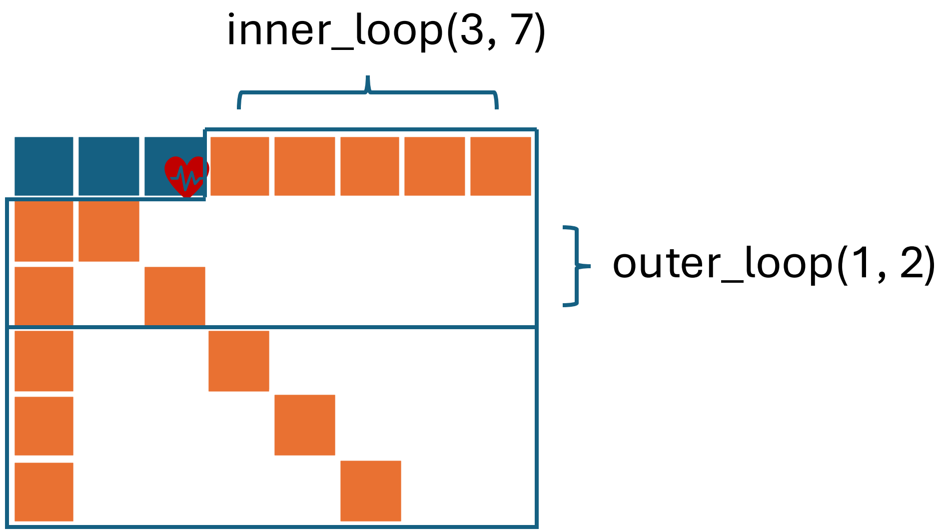
```
#pragma omp parallel for
for (int i = 0; i < M.num_rows(); i++) {
  initialWork();
  #pragma omp parallel for
  for (int j = 0; j < M.num_nonzeros(i); j++) {
    processElement(M, i, j);
  }
  writeResult();
}
```

```
void outer_loop(int startIter, int endIter) {
  for (; startIter < endIter; startIter++) {
    initialWork();
    inner_loop(0, M.num_nonzeros(startIter));
    writeResult();
  }
}
```

```
void inner_loop(int startIter, int endIter) {
  for (; startIter < endIter; startIter++) {
    processElement(M, i, startIter);
  }
}
```

```
void outer_loop(int startIter, int endIter) {
  for (; startIter < endIter; startIter++) {
    initialWork();
    inner_loop(0, M.num_nonzeros(startIter));
    writeResult();
  }
}
```

```
void inner_loop(int startIter, int endIter) {
  for (; startIter < endIter; startIter++) {
    processElement(M, i, startIter);
  }
}
```



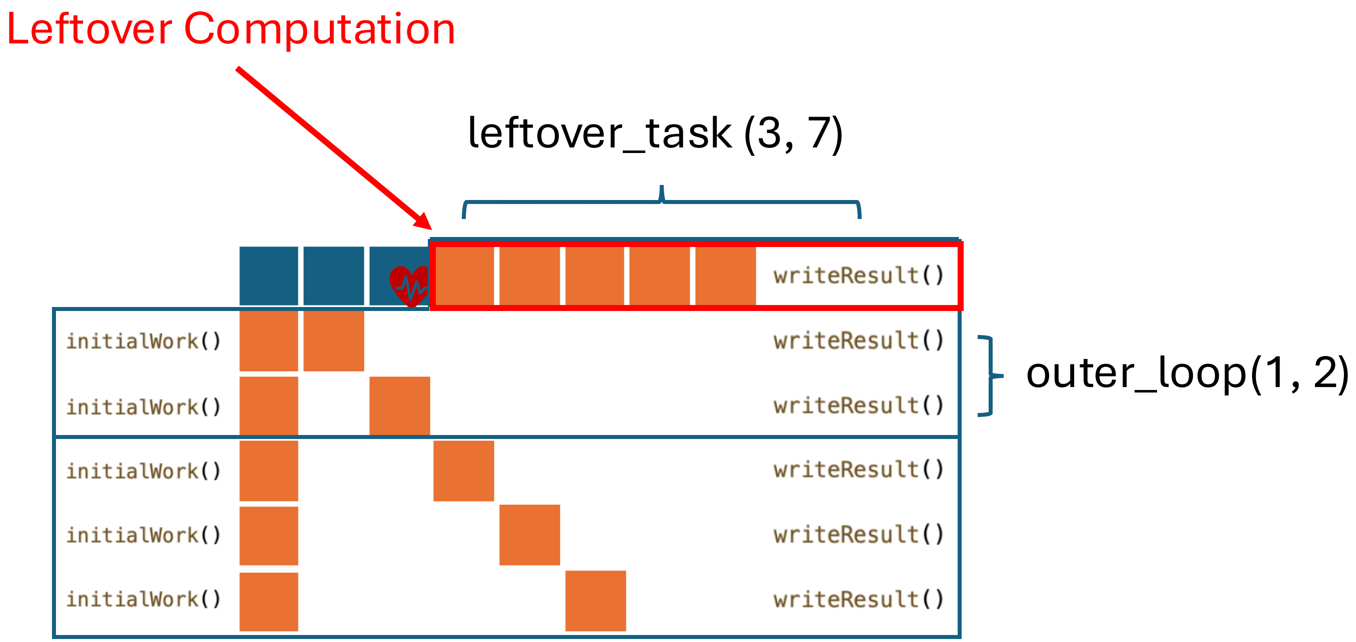
```
void outer_loop(int startIter, int endIter) {
  for (; startIter < endIter; startIter++) {
    initialWork();
    inner_loop(0, M.num_nonzeros(startIter));
    writeResult();
  }
}
```

```
void inner_loop(int startIter, int endIter) {
  for (; startIter < endIter; startIter++) {
    processElement(M, i, startIter);
  }
}
```

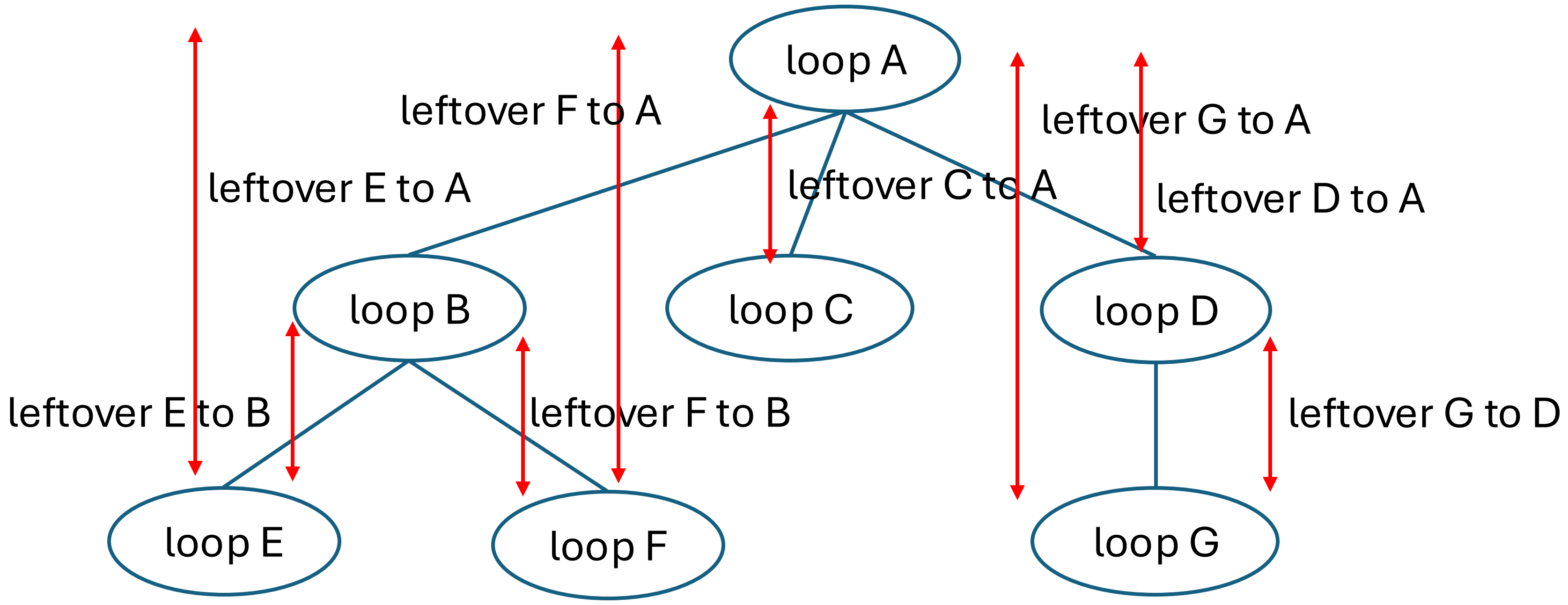
```
void outer_loop(int startIter, int endIter) {
  for (; startIter < endIter; startIter++) {
    initialWork();
    inner_loop(0, M.num_nonzeros(startIter));
    writeResult();
  }
}
```

```
void inner_loop(int startIter, int endIter) {
  for (; startIter < endIter; startIter++) {
    processElement(M, i, startIter);
  }
}
```

```
void leftover_task(int startIter, int endIter) {
  inner_loop(startIter, endIter)
  writeResult();
}
```



HBC automates leftover task generation

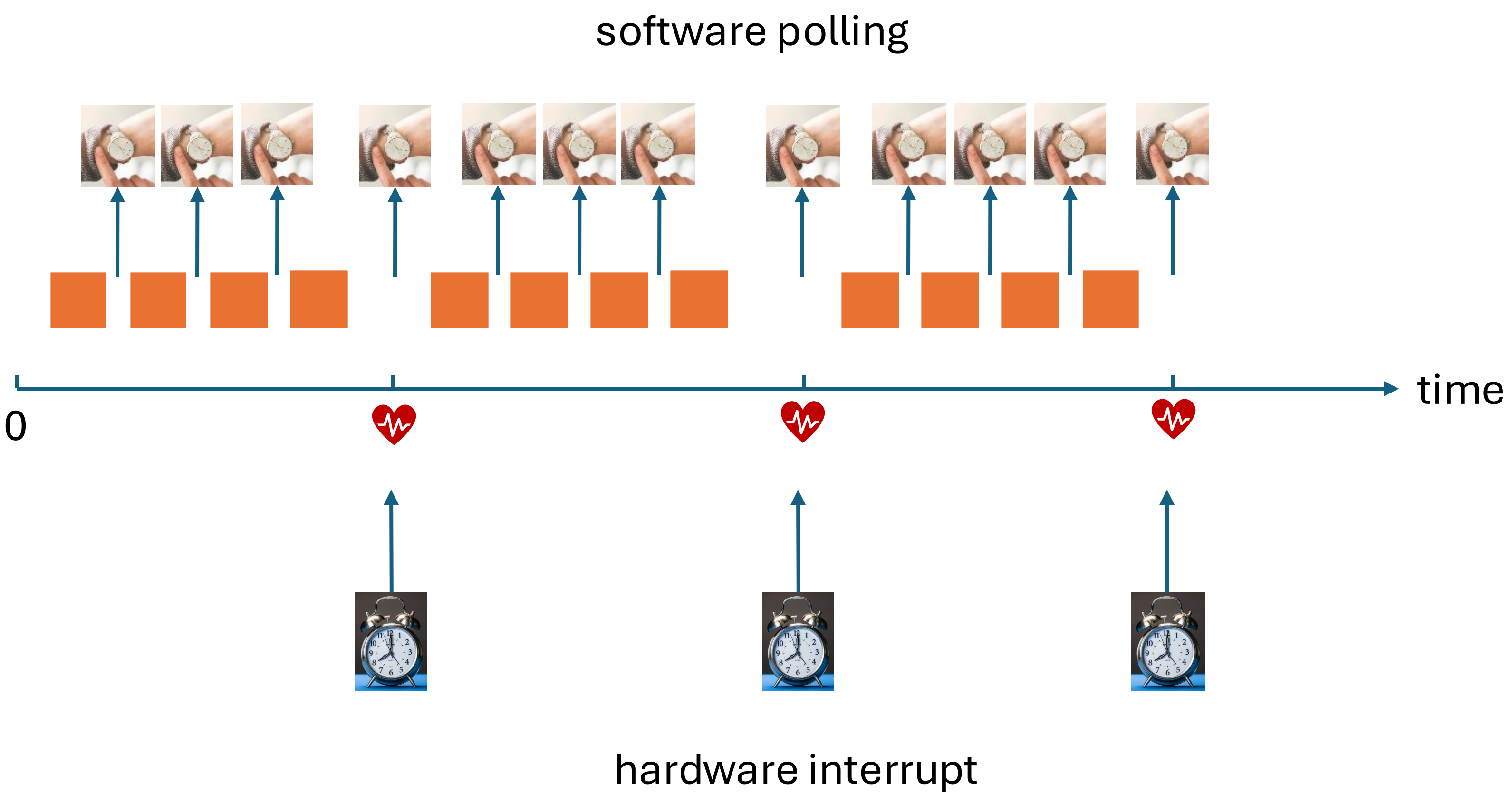


A 3-level loop nesting tree with 7 parallel-for loops

Nesting tree depth: $\boldsymbol{D}$
Maximum sibling loops among all levels: $\boldsymbol{s_{max}}$
Total leftover tasks: $O(D^2 \times s_{max})$

HBC ✔

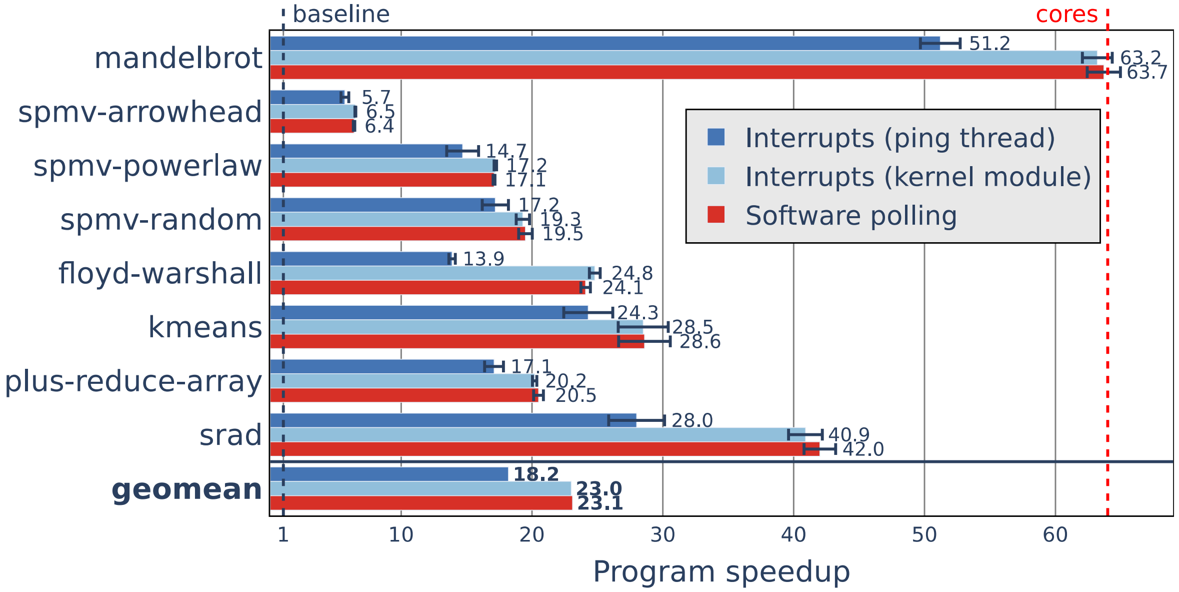
Heartbeat Delivery



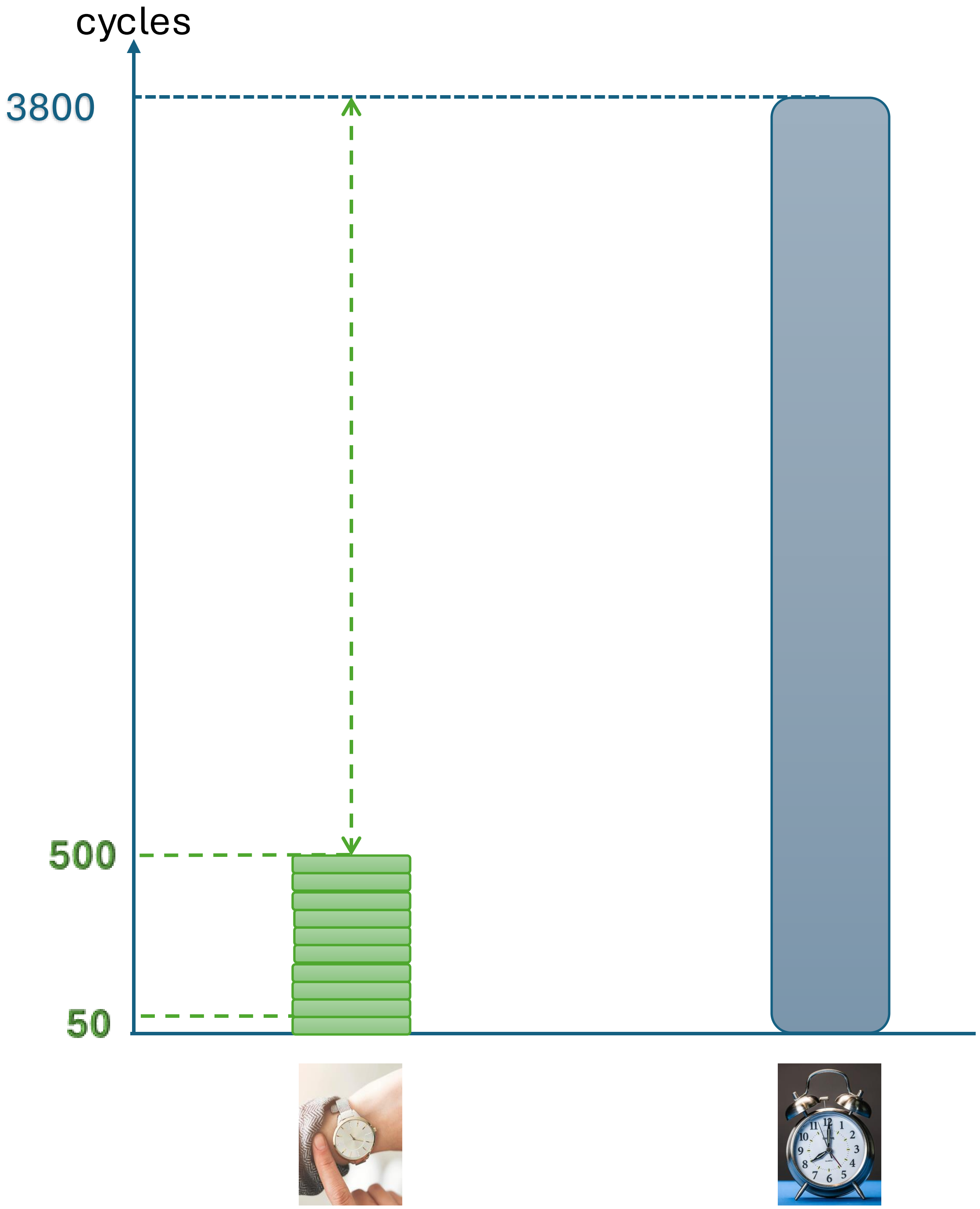
hardware interrupt

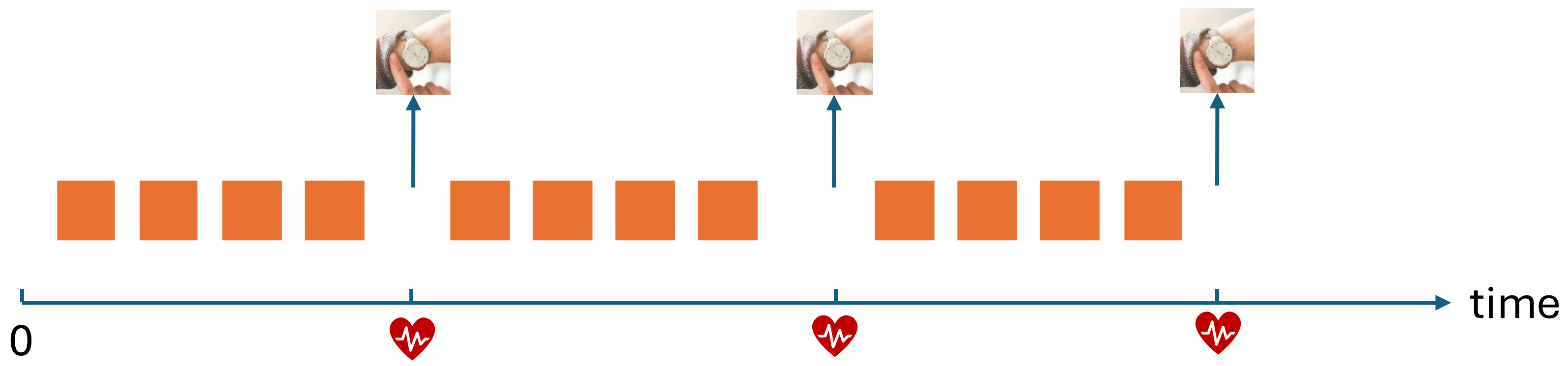
v.s.

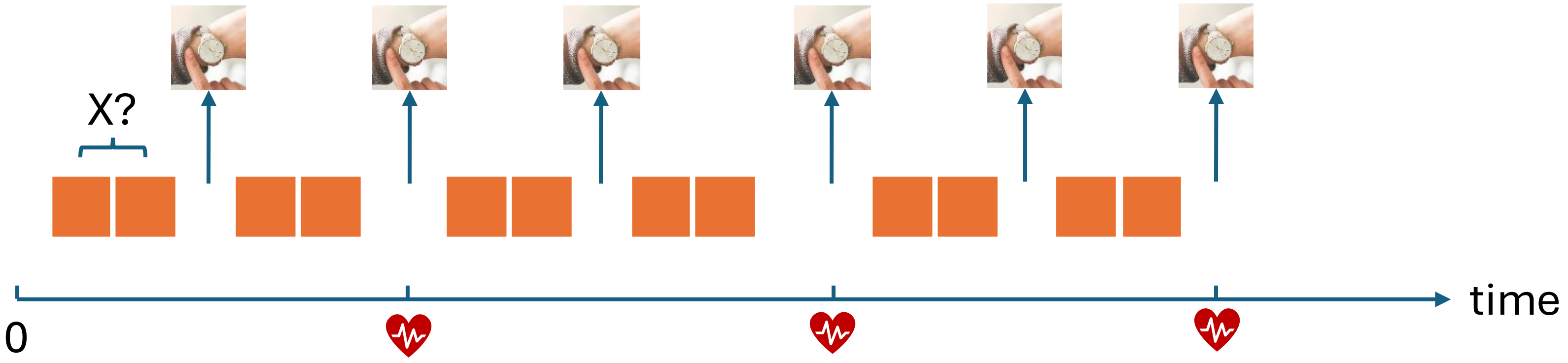
software polling



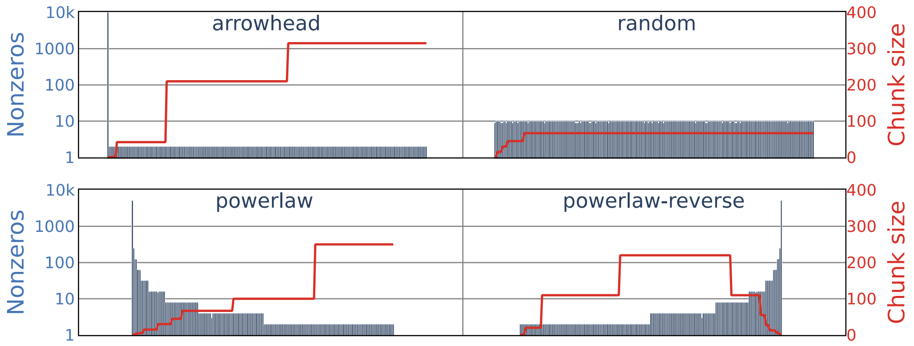
software polling achieves the same performance against a state-of-the-art, dedicated kernel module for heartbeat delivery!


cycles
3800
500
50


time
0



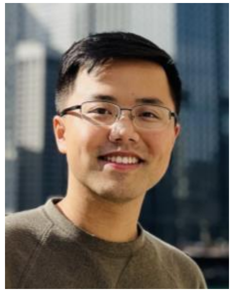
Adaptive Chunking (AC)



hardware interrupt

software polling

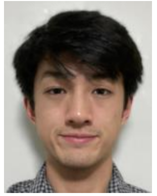
Compiling Loop-Based Nested Parallelism for Irregular Workloads



Yian Su

Mike Rainey

Nick Wanninger



Nadharm Dhiantravan

Jasper Liang

Umut A. Acar

Peter Dinda

Simone Campanoni

yiansu2018@u.northwestern.edu

yiansu.com

ARCANA Rethinks Compiler Abstractions for New Applications

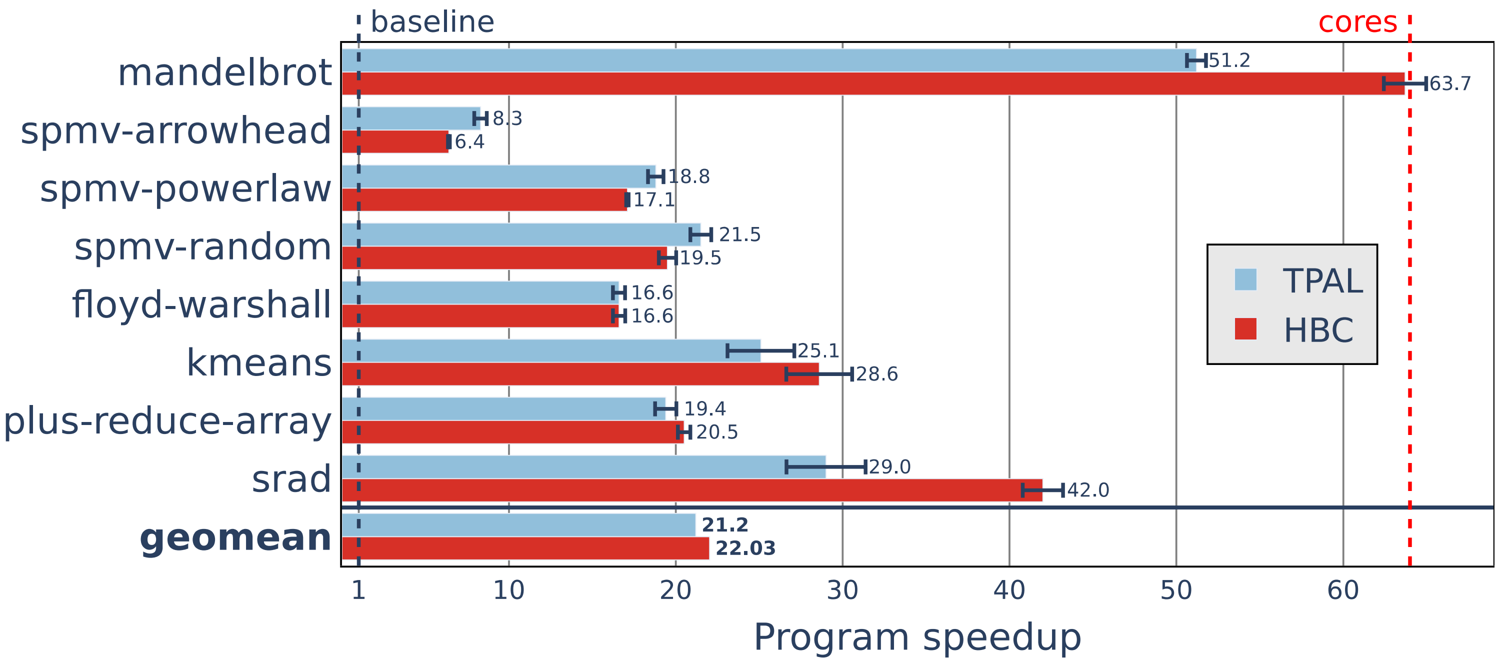
Northwestern University

Carnegie Mellon University

Benchmarks

Benchmark	Source	Input	Regularity
OpenMP pragmas are generated by programmers			
mandelbrot	TPAL [40, 42]	512 × 1024 × 40k	irregular
spmv-arrowhead	TPAL [40, 42]	150 million rows 450 million nonzeros	irregular
spmv-powerlaw	TPAL [40, 42]	16.7 million rows 402 million nonzeros	irregular
spmv-random	TPAL [40, 42]	6 million rows 600 million nonzeros	regular
floyd-warshall	TPAL [40, 42]	4k × 4k	regular
kmeans	TPAL [40, 42]	10 million elements	regular
plus-reduce-array	TPAL [40, 42]	100 billion elements	regular
srad	TPAL [40, 42]	10k × 10k	regular
mandelbulb	3D Mandelbrot [52]	100 × 200 × 300 × 400	irregular
cg	NAS [39]	cage15 [50]	irregular
OpenMP pragmas are automatically generated			
ttv	TACO [28, 29]	nell-2 [46]	irregular
ttm	TACO [28, 29]	nell-2 [46]	irregular
bfs	GraphIt [54, 55]	Twitter [30]	irregular
cc	GraphIt [54, 55]	Twitter [30]	irregular
pr	GraphIt [54, 55]	Twitter [30]	irregular
cf	GraphIt [54, 55]	LiveJournal [12]	irregular
pr-delta	GraphIt [54, 55]	LiveJournal [12]	irregular
sssp	GraphIt [54, 55]	LiveJournal [12]	irregular

HBC vs Manual Implementation (TPAL[1])



HBC delivers comparable performance against the state-of-the-art manual implementation of heartbeat scheduling!

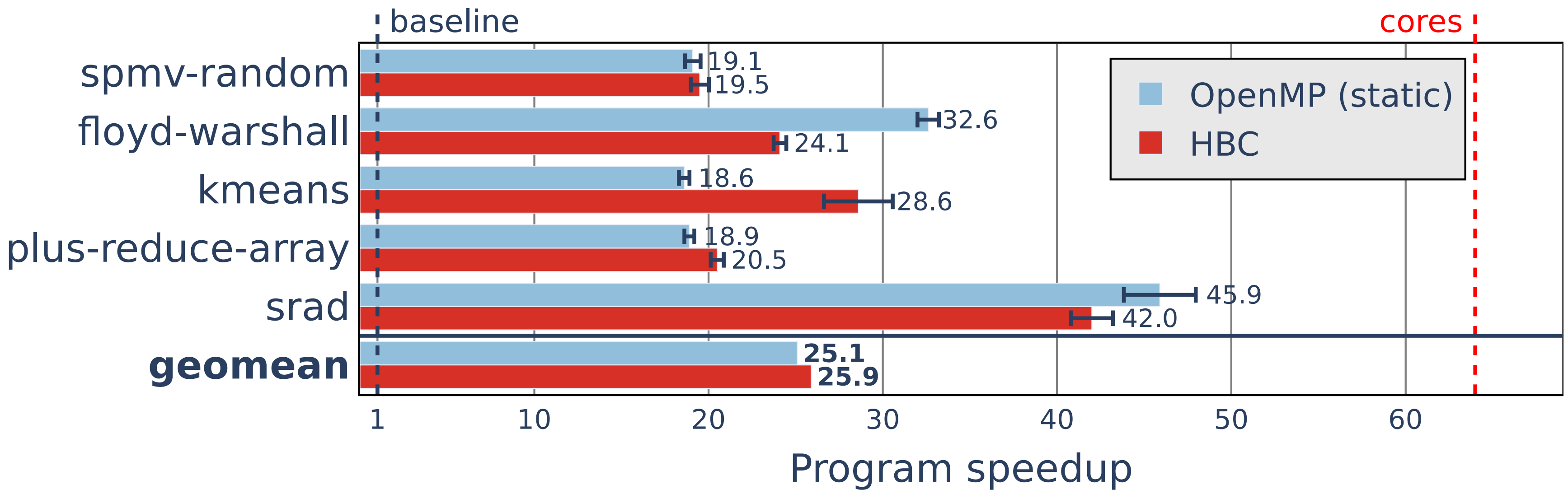
[1] Mike Rainey, Ryan R. Newton, Kyle Hale, Nikos Hardavellas, Simone Campanoni, Peter Dinda, and Umut A. Acar. *Task parallel assembly language for uncompromising parallelism*. PLDI 2021

Use chunk size dynamically

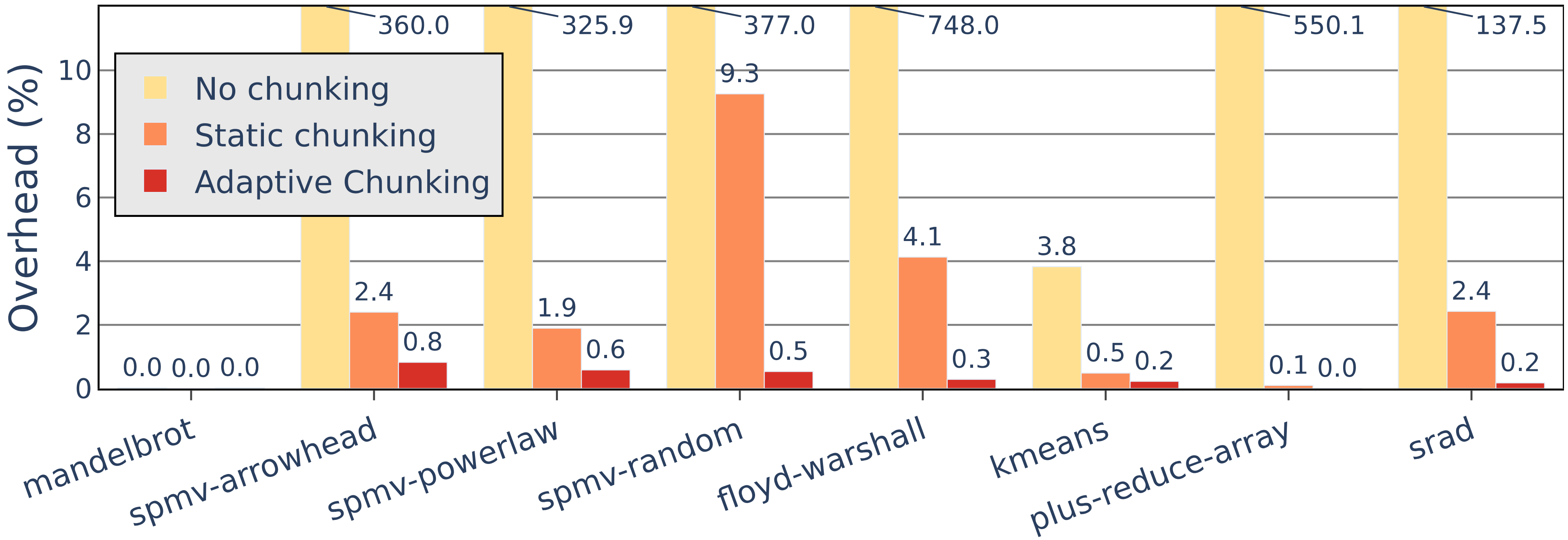
```
int chunk_size;
for (; startIter < maxIter; startIter += chunk_size) {
  chunk_size = get_chunk_size();

  int low = startIter;
  int high = startIter+chunksize > maxIter ? maxIter : startIter+chunksize;
  for (; low < high; low++) {
    // loop_body();
  }
}
```

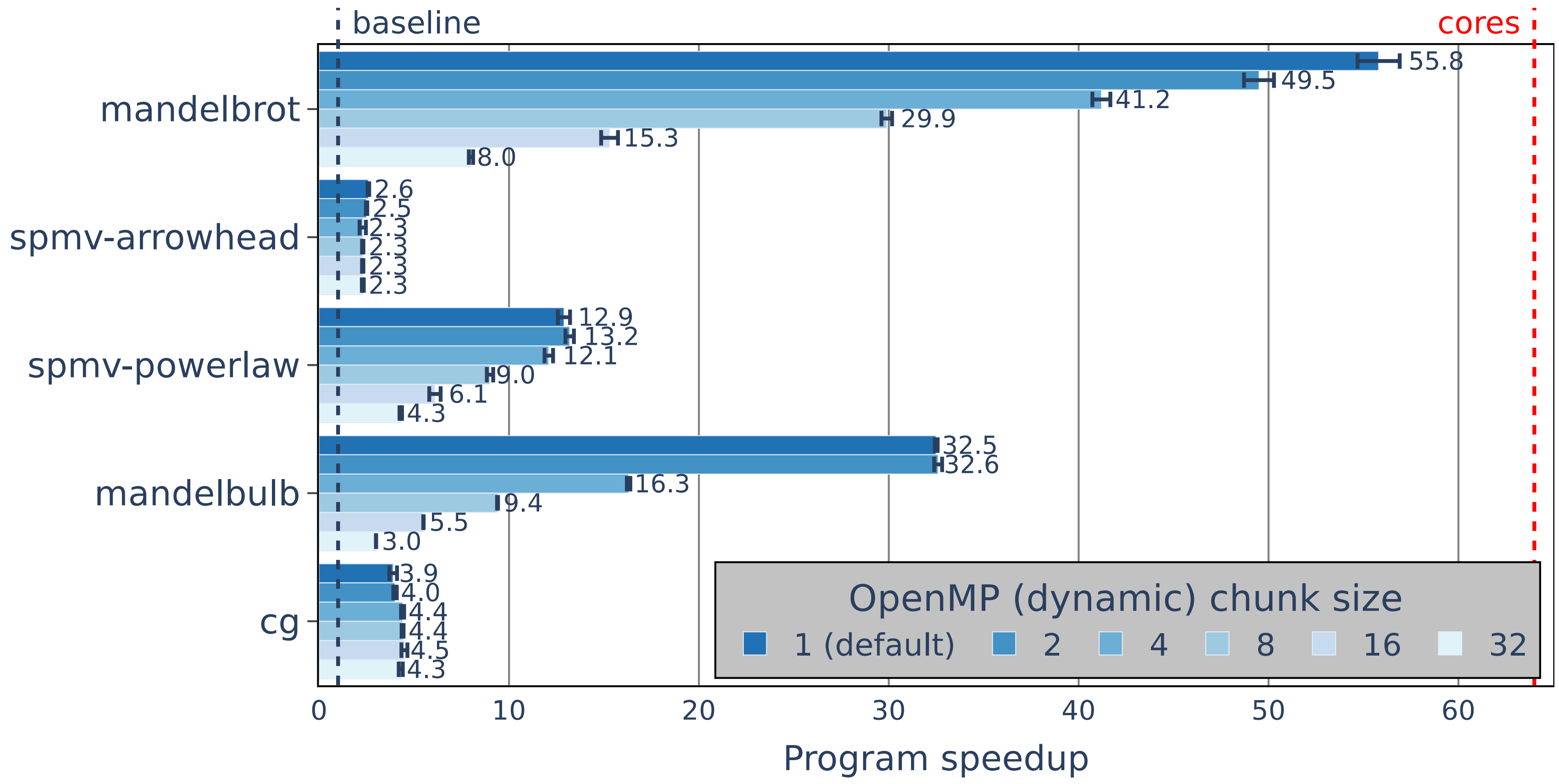
HBC for Regular Workloads



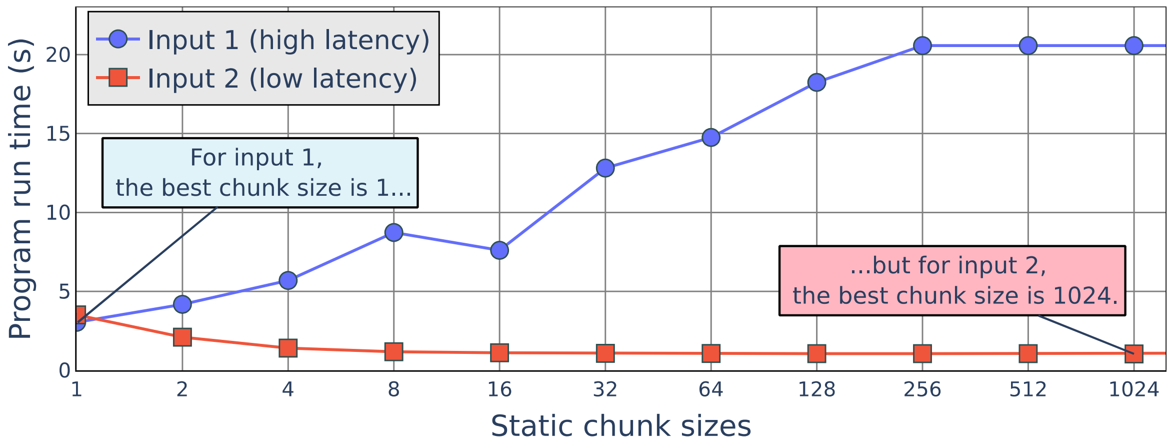
Software Polling Overhead

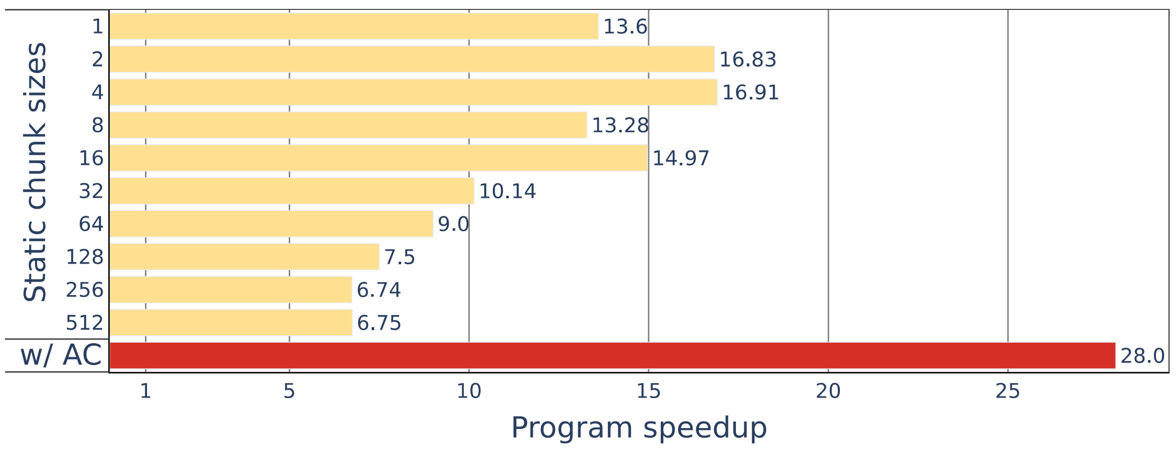


Tuning OpenMP chunk size



Why Adaptive Chunksize?





Heartbeat Detection