PCERE: Fine-grained Parallel Benchmark Decomposition for Scalability Prediction

Mihail Popov, Chadi Akel, Florent Conti, William Jalby,
Pablo de Oliveira Castro

UVSQ - PRiSM - ECR

Mai 28, 2015

Evaluate strong scalability

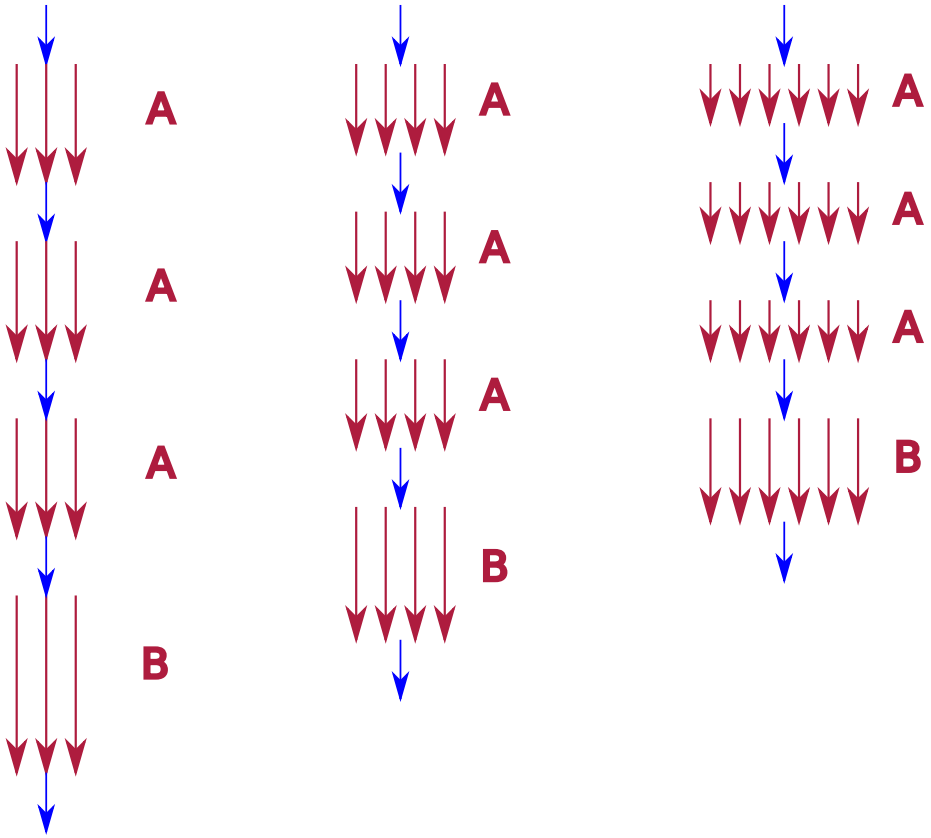
- Evaluate strong scalability of OpenMP applications is costly and time-consuming
- Execute multiple times the whole application with different thread configurations
- Waste of ressources
  - According to Amdahl's law sequential parts do not scale
  - Parallel regions may share similar performance across invocations

PCERE: Parallel Codelet Extractor and REplayer

- *Accelerate strong scalability evaluation* with PCERE
- PCERE is part of CERE (Codelet Extractor and REplayer) framework
- Decompose applications into small pieces called *Codelets*
- Each codelet maps a parallel region and is a standalone executable
- Extract codelets once
- Replay codelets instead of applications with different number of threads

Prediction model

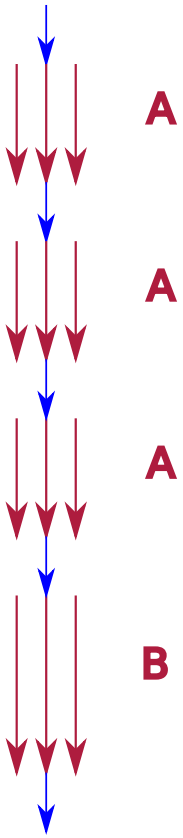
```
int main()
{
for(i=0;i<3;i++){
  //sequentiel code
  #pragma omp parallel A
}
//sequentiel code
#pragma omp parallel B
//sequentiel code
}
```



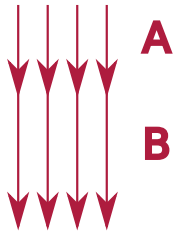
Executing the whole application with different threads configurations

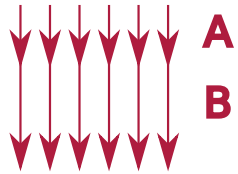
Prediction model

```
int main()
{
for(i=0;i<3;i++){
  //sequentiel code
  #pragma omp parallel A
}
//sequentiel code
#pragma omp parallel B
//sequentiel code
}
```



Extracting parallel regions A and B and measuring sequentiel execution time

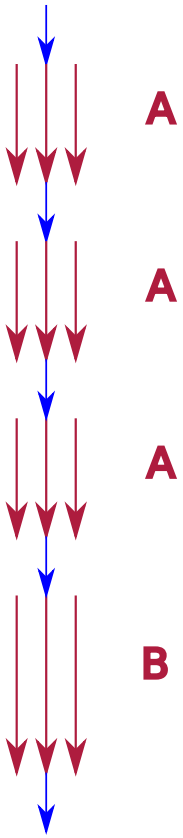


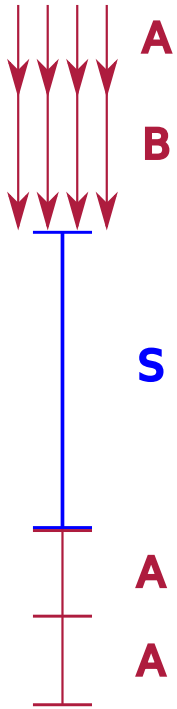


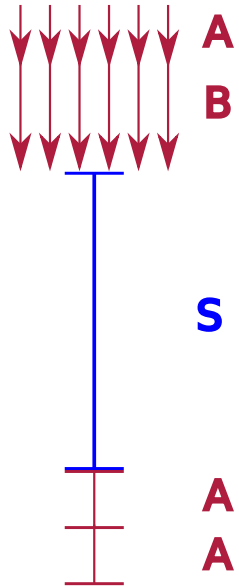
Directly replaying the parallel regions

Prediction model

```
int main()
{
for(i=0;i<3;i++){
  //sequentiel code
  #pragma omp parallel A
}
//sequentiel code
#pragma omp parallel B
//sequentiel code
}
```



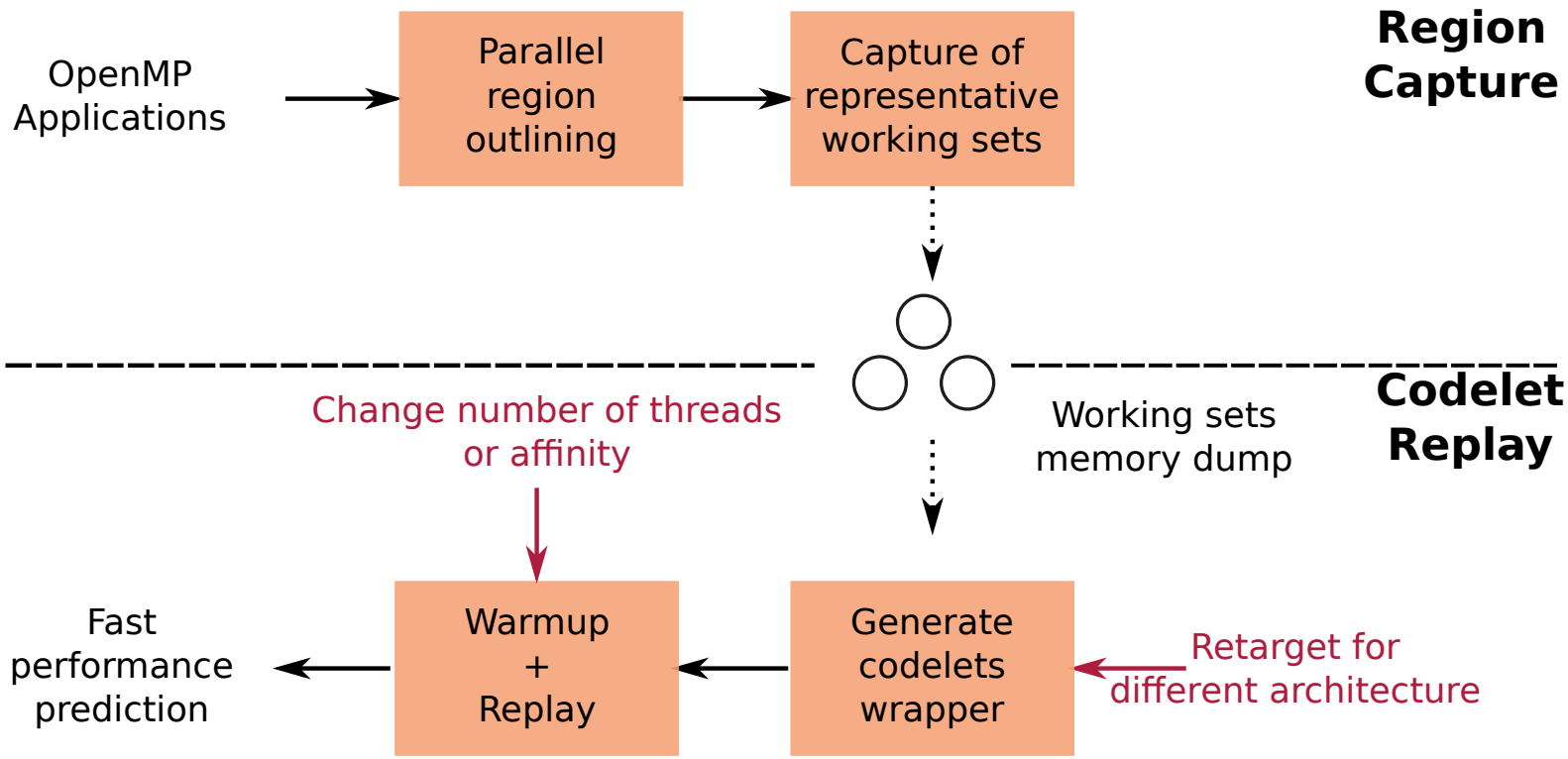




Add sequential time
and
parallel region multiple invocations

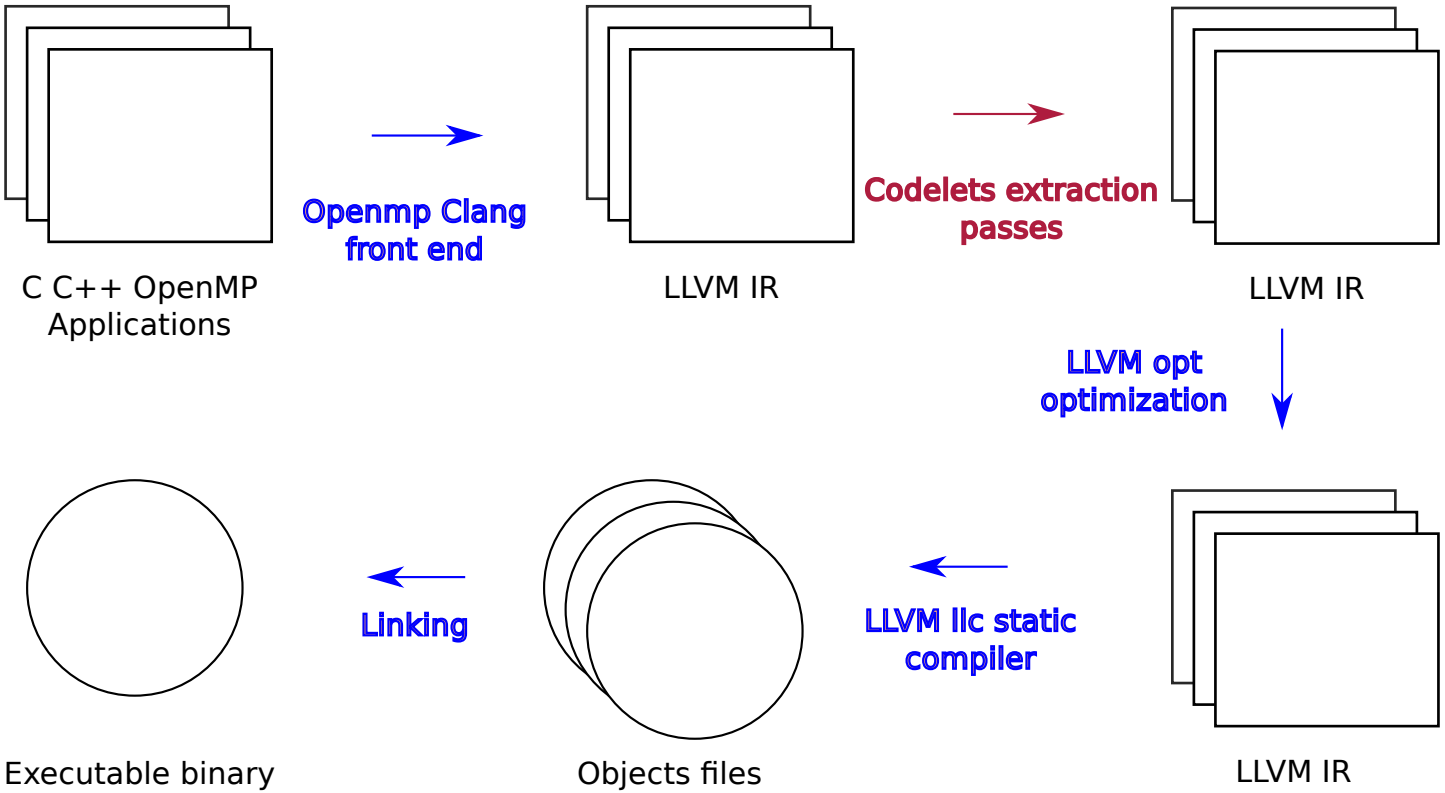
Outline

Codelet capture and replay

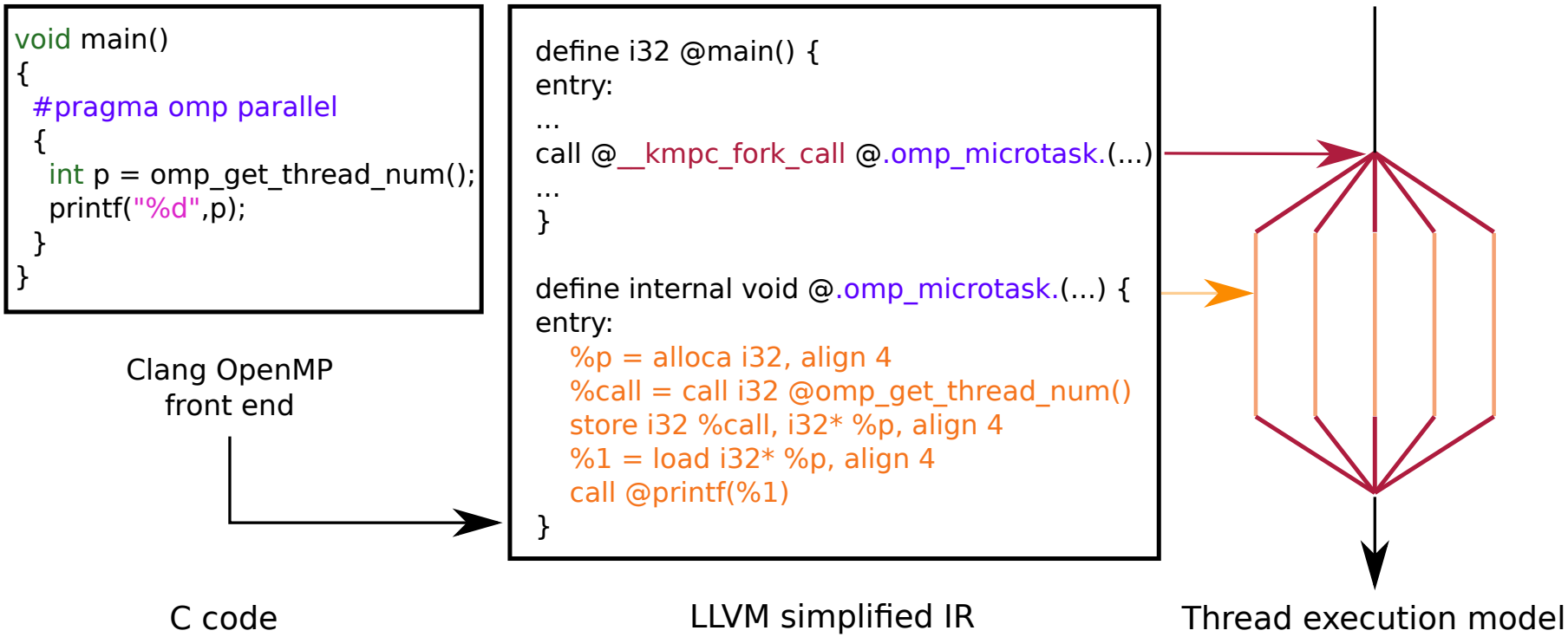


LLVM OpenMP Intermediate Representation extraction

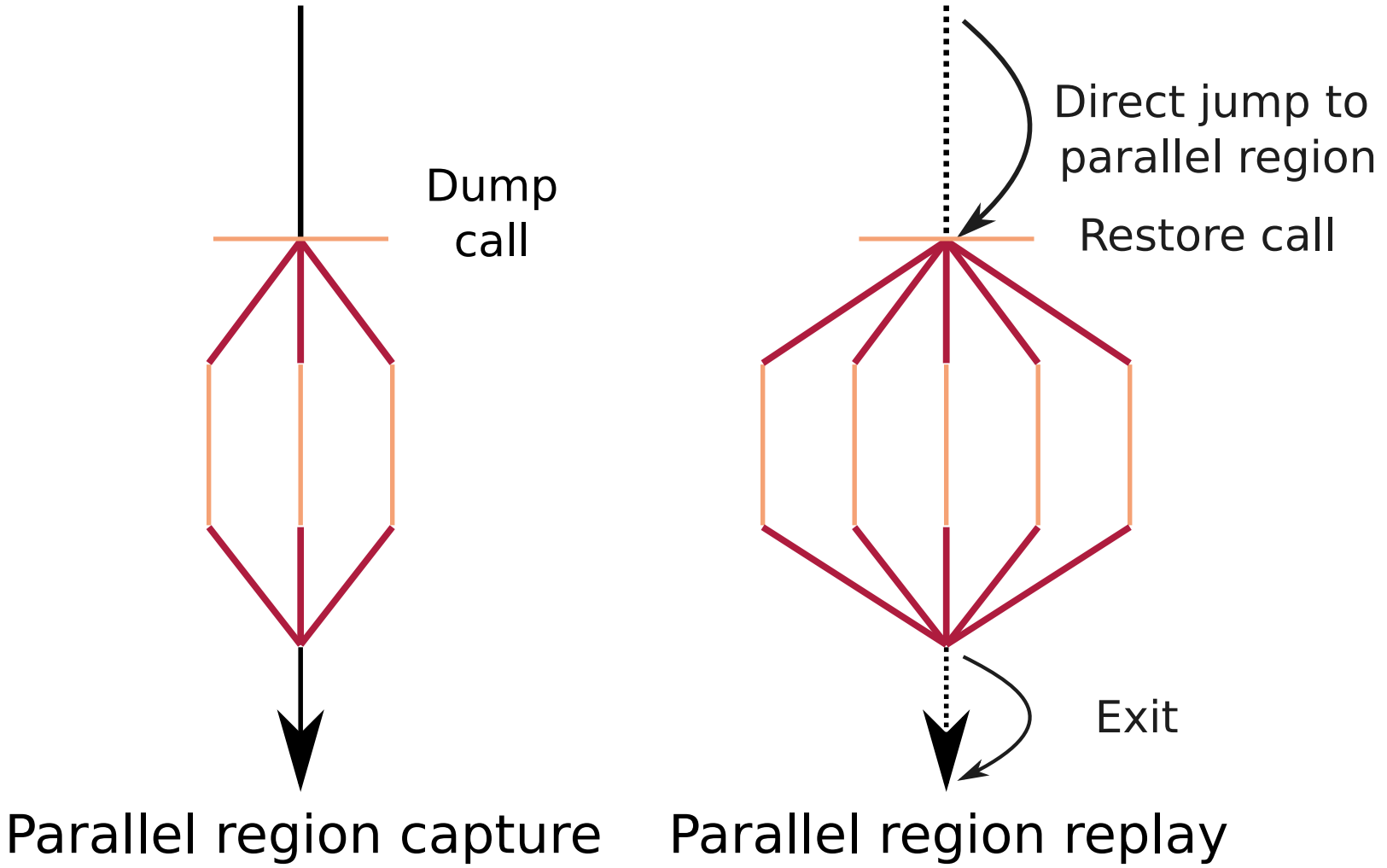
- Extract codelets at Intermediate Representation for language portability and cross architecture evaluation



Clang front end transforms source code into IR



Deterministic codelet replay

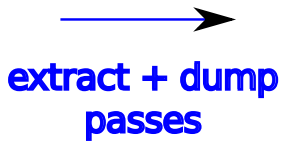


Memory dump

- System memory snapshot at the beginning of each parallel region

```
define i32 @main() {
entry:
...
call @__kmpc_fork_call @.omp_microtask.(...)
...
}

define internal void @.omp_microtask.(...) {
entry:
  %p = alloca i32, align 4
  %call = call i32 @omp_get_thread_num()
  store i32 %call, i32* %p, align 4
  %1 = load i32* %p, align 4
  call @printf(%1)
}
```

LLVM simplified IR




```
define i32 @main() {
entry:
...
call @__extracted__.omp_microtask.(...)
...
}

define internal void @__extracted__.omp_microtask.(...){
newFuncRoot:
call void @dump(...)
call @__kmpc_fork_call @ .omp_microtask.(...)
}

define internal void @.omp_microtask.(...) {
entry:
  ...
}
```

LLVM simplified IR

Codelet replay

- Reload codelet working set
- Reproduce cache state with optimistic cache warm-up
- Multiple working sets for a single codelet

Codelets with different working sets

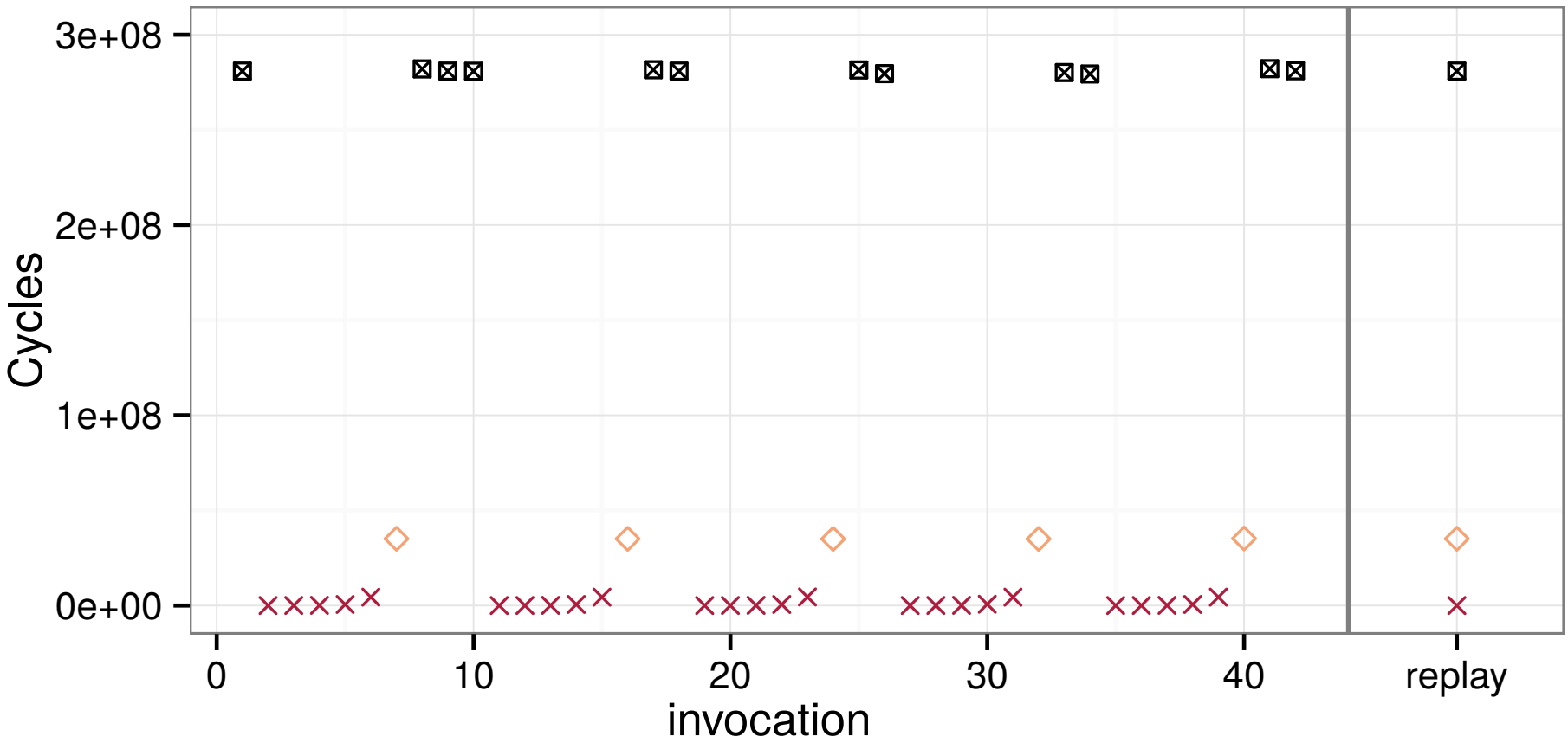


Figure : MG resid execution time over the different invocations replayed with 4 threads

Lock Support

- Lock support on Linux uses Futex
- Each futex allocates a kernel space wait queue
- Memory capture saves only the user space memory
- Lock capture step that detects all the locks accessed by a codelet
- Replay wrapper initialize required locks in kernel space

Test benchmarks and architectures

- Using NAS Parallel Benchmark OpenMP 3.0 C version based on the Omni Compiler Project

	Core2	Nehalem	Sandy Bridge	Ivy Bridge
CPU	E7500	Xeon E5620	E5	i7-3770
Frequency (GHz)	2.93	2.40	2.7	3.4
Sockets	1	2	2	1
Cores per socket	2	4	8	4
Threads per core	1	1	2	2
L1 cache (KB)	32	32	32	32
L2 cache (KB)	3MB	256	256	256
L3 cache (MB)	-	12	20	8
Ram (GB)	4	24	64	32

Figure : Test architectures

Reproducing parallel regions scaling with codelets

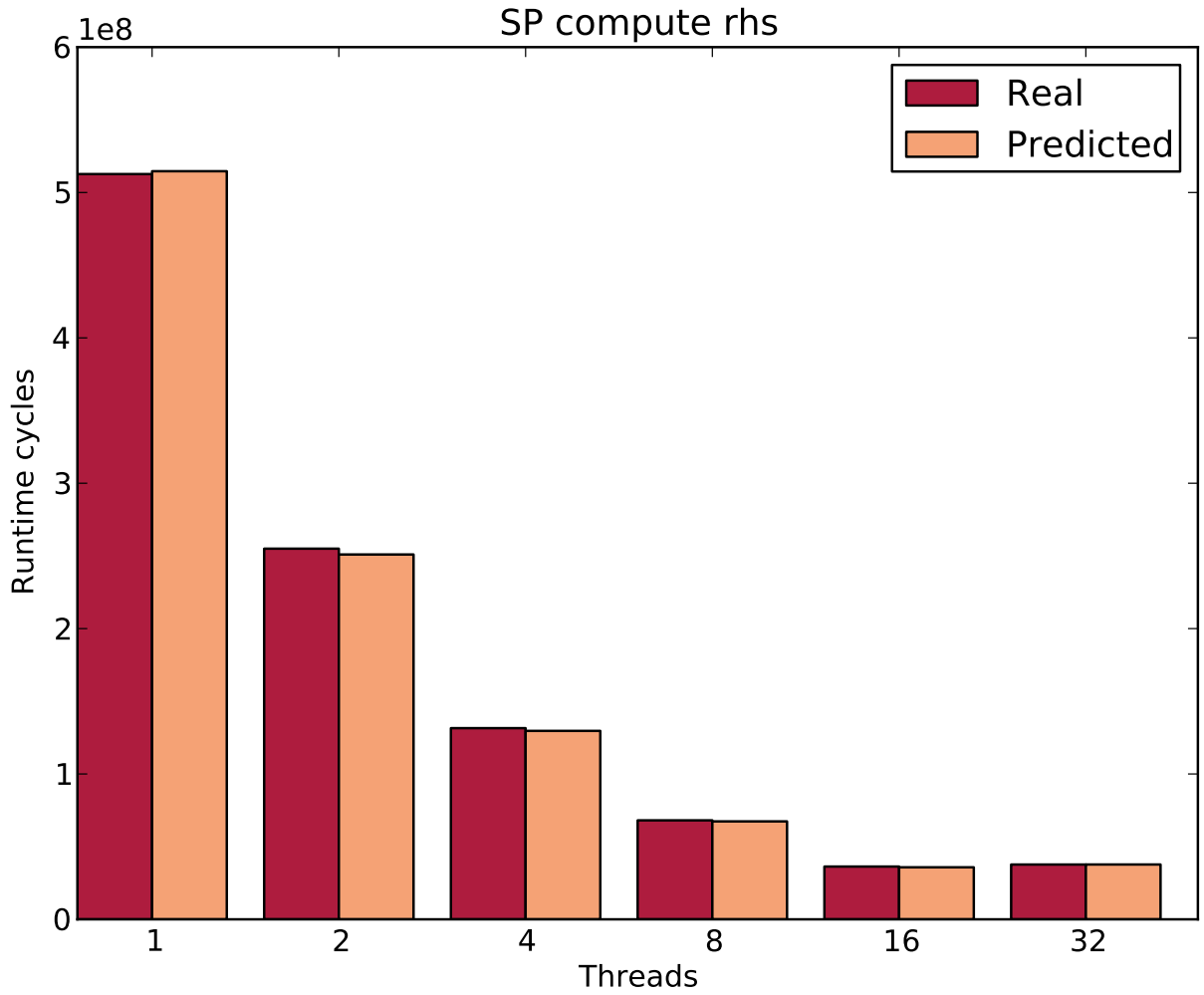


Figure : Real vs. PCERE execution time predictions on Sandy Bridge for the SP compute rhs codelet

Prediction accuracy

	BT	EP	LU	FT	SP	CG	IS	MG
Core2	2.4	0.1	1.4	2.3	1.8	1	4.2	1.5
Nehalem	3	0.4	0.6	0.5	6	9.8	2.2	0.6
Sandy Bridge	8.9	1.5	1.4	1.6	0.9	14.5	18	12.1
Ivy Bridge	0.7	1	2	3.4	1.2	3.7	5.3	5.4

Figure : NAS 3.0 C version **average percentage error** prediction accuracy

- *On Ivy Bridge, PCERE predicts FT execution time scalability with an error of 3.4%*

Benchmarking acceleration

	BT	EP	LU	FT	SP	CG	IS	MG
Core2	31.5	1	54.3	1.5	87.2	24.2	1.1	0.9
Nehalem	43.2	1	51.9	2.1	97	21.1	1.2	2.2
Sandy Bridge	45.5	1	44.6	2.4	79	13.2	1.2	2.4
Ivy Bridge	39	1	45.5	2.1	82	17	1.1	1.8

Figure : NAS 3.0 C version **average benchmarking acceleration**

- *On Core2, PCERE CG scalability evaluation is 24.2 times faster than with normal executions*

PCERE prediction accuracy and benchmarking acceleration

	Core2	Nehalem	Sandy Bridge	Ivy Bridge
Accuracy	1.8%	2.9%	7.4%	2.8%
Acceleration	25.2	27.4	23.7	23.7

Figure : NAS 3.0 C version average **prediction accuracy** and **benchmarking acceleration** per architecture

Cross micro-architecture codelet replay

- Capture-Replay is micro-architecture agnostic
- Capture on Nehalem → Replay on Sandy Bridge

Threads	1	2	4	8	16	32
Accuracy	3	2.3	3	7.8	11.5	17.6

Figure : NAS 3.0 C version **average percentage error** cross replay accuracy

Application	BT	EP	LU	FT	SP	CG	IS	MG
Accuracy	9.8	0.3	1.1	3.8	3.9	18.1	6.4	17

Figure : NAS 3.0 C version **average percentage error** cross replay accuracy

Limitation and future work

- Limitations
  - No acceleration on applications with a single parallel region and no relevant sequential parts (EP)
  - Prediction error due to variant sequential time across thread configurations (IS)
- Future work
  - Improve warm-up strategy: use CERE page traces warm-up
  - Apply a clustering approach over codelets
  - OpenMP parameters space exploration with codelets

Conclusion

- To be released with CERE at *http://benchmark-subsetting.github.io/pcere/*
- Extract codelets once, replay them many times
- Cross micro-architecture and thread configuration extraction and replay
- Accelerate strong scalability evaluation 25 times
- Strong scalability prediction average error of 3.7%

Codelet replay

- Optimistic cache warm-up: assuming that the codelet working set is hot in the original run

extract + replay passes

```
void main()
{
  int i;
  int iteration = 1;
  for(i=0;i<iteration;i++)
    run__extracted__.omp_microtask.();
}
```

Updated main C code

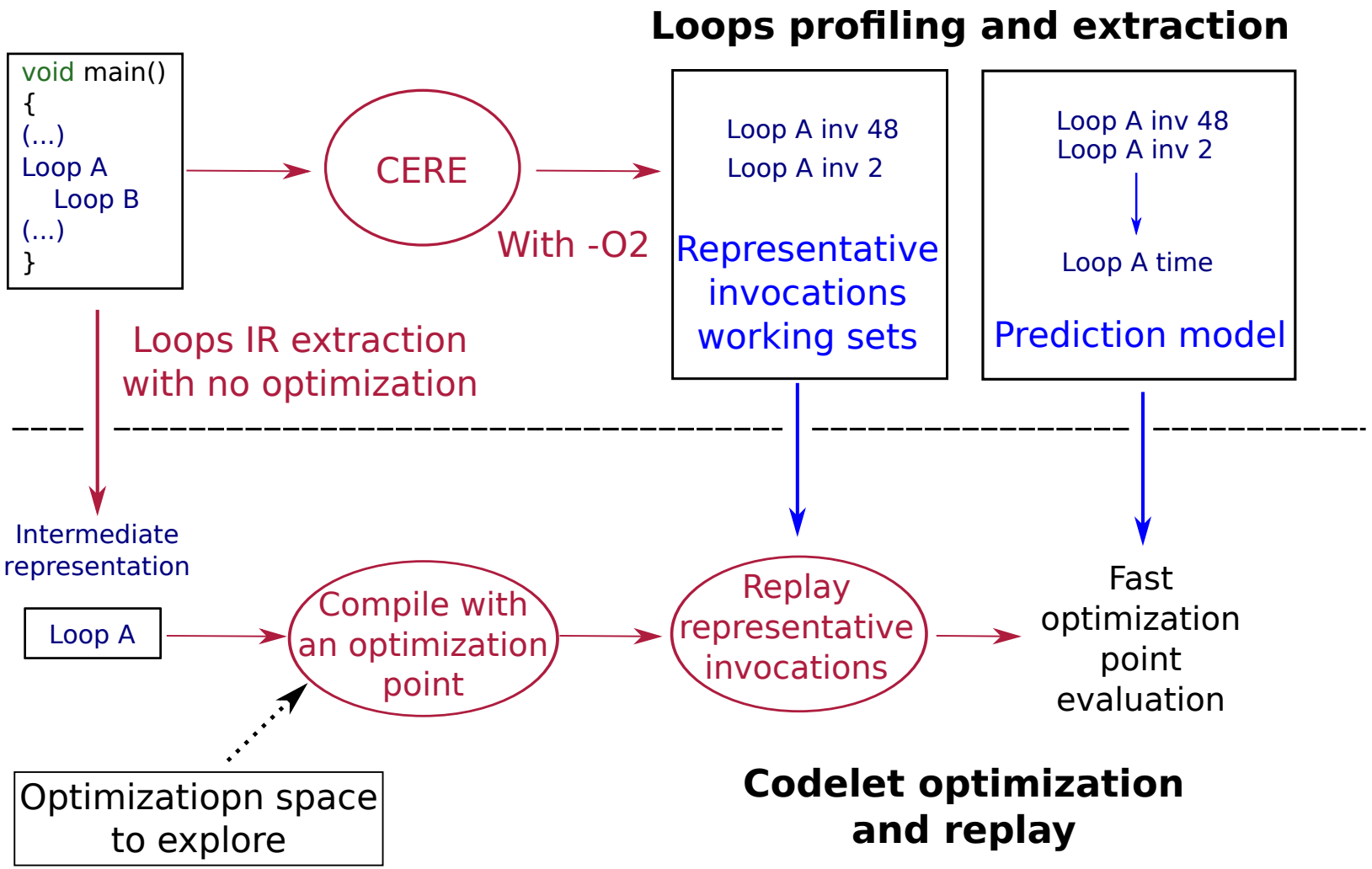
```
define void @run__extracted__.omp_microtask.() {
entry:
call void @load(...)
... %Arrange arguments
call @__extracted__.omp_microtask.(...)
}

define internal void @__extracted__.omp_microtask.(...){
newFuncRoot:
call @__kmpc_fork_call @ .omp_microtask.(...)
}

define internal void @.omp_microtask.(...) {
entry:
  ...
}
```

LLVM simplified IR

Related work

- Cross-platform performance prediction of parallel applications using partial execution. Yang, Leo T and Ma, Xiaosong and Mueller, Frank SC 2005
- Detecting Phases in Parallel Applications on Shared Memory Architectures. Perelman, Erez and Polito, Marzia and Bouguet, J-Y and Sampson, Jack and Calder, Brad and Dulong, Carole IPDPS 2006
- BarrierPoint: Sampled Simulation of Multi-Threaded Applications. Carlson, Trevor E and Heirman, Wim and Van Craeynest, Kenzo and Eeckhout, Lieven ISPASS 2014
- Effective source-to-source outlining to support whole program empirical optimization. Liao, Chunhua and Quinlan, Daniel J and Vuduc, Richard and Panas, Thomas Languages and Compilers for Parallel Computing 2010

Flags exploration



Flags exploration

- For each optimization sequence, only replay the relevant parts
- Codelets matching over 200 optimization sequences

Application	Median error	Average error
CG	2.0	3.0
EP	0.1	0.3
FT	3.6	5.0
IS	1.3	4.8
LU	1.2	2.2
MG	1.9	3.4
SP	2.1	2.6
RTM	5.3	5.8

Figure : Matching **error percentage** per application

- Speed-up evaluation versus matching error
  - -O2 RTM evaluation is 237 times cheaper with codelets