Software Behavior Oriented Parallelization

Xipeng Shen

The College of William and Mary

Joint work with Chen Ding, Kirk Kelsey, Chris Tice, Ruke Huang, and Chengliang Zhang

at University of Rochester and Microsoft

High-level Parallelism

- Parallel computing is becoming ubiquitous
- High-level parallelism exists in many programs
  - E.g. utilities, interpreters, scientific computations
  - Difficult to parallelize

Complexity in the code

Bit-level operations,
unrestricted pointers,
exception handling,
custom mem. management,
third-party libraries

Uncertain parallelism

Example*:

```
while ( s=nextSentence() )
{  parse(s);
   if ( isCommand(s) )
       updateParsingEnv(s);
}
```

* Simplified Parser in SPEC2k by Sleator & Temperly

Behavior Oriented Parallelization (BOP)

- Goal
  - To improve executions with coarse-grained uncertain parallelism, while guaranteeing correctness and basic efficiency.
- Components
  - Selection of possibly parallel regions (PPRs)
    - By the programmer or a profiling tool
  - Data & code transformation
    - By a compiler
  - Runtime protection of correctness & efficiency
    - By a runtime system

Possibly Parallel Regions (PPRs)

```
while (1) {
  get_work( );
  . . .
  BeginPPR(1);
  step1( );
  step2( );
  EndPPR(1);
  . . .
}
```

```
. . .
BeginPPR(1);
work(x);
EndPPR(1);
. . .
BeginPPR(2);
work(y);
EndPPR(2);
. . .
```

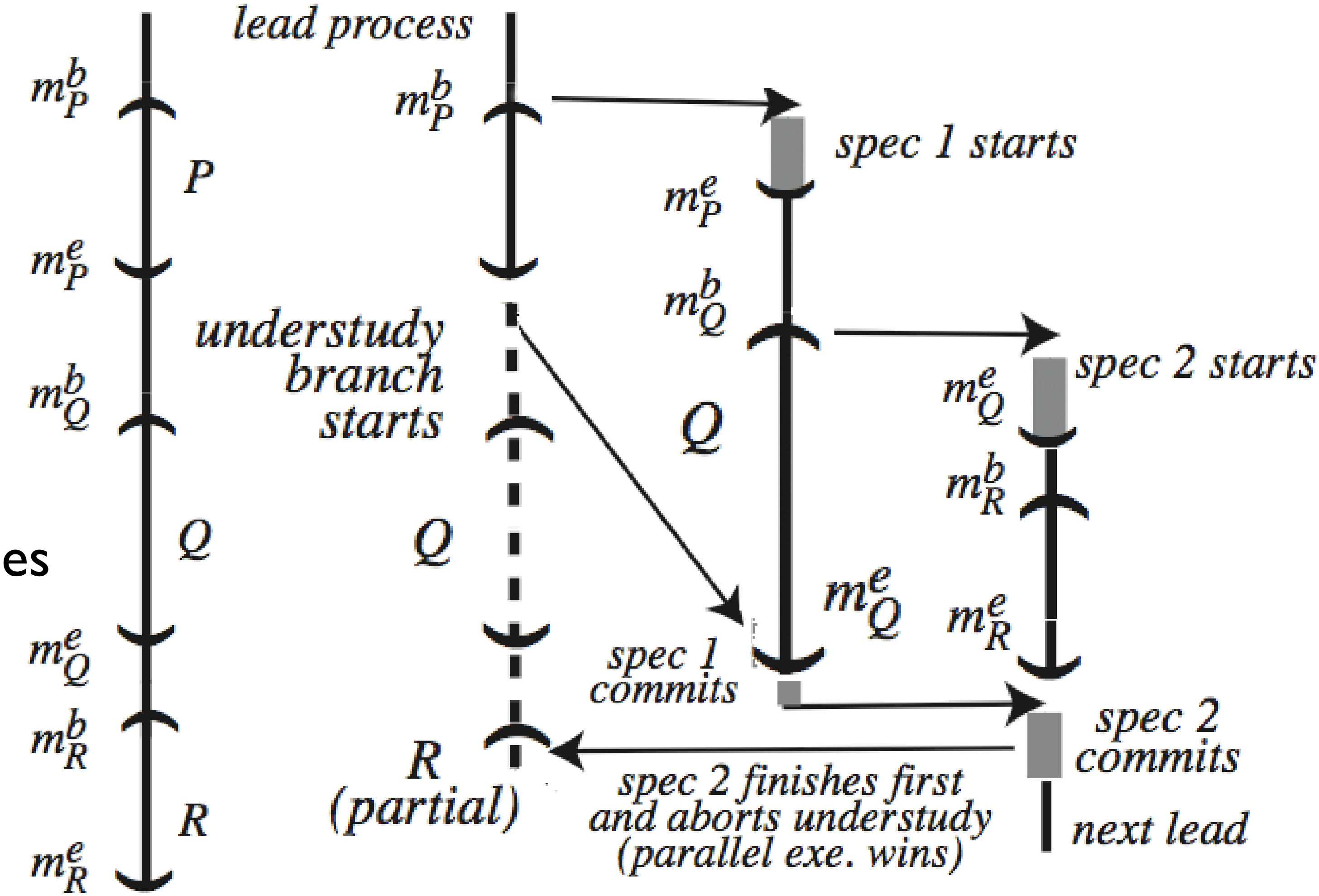
- Region-based
- Likely parallelism
- Allows unpredictable entries or exits

At BeginPPR, a speculation process is created, which jumps to EndPPR and starts executing there.

Just hints, no harm to correctness

BOP Execution Model

A Prog. with 3 PPR instances



BOP Execution Model

- Process-level parallelization
  - Strong isolation of processes
- Overhead is addressed by
  - forming a speculation pipeline with overhead overlapped
  - keeping most overhead off the critical path
  - using understudy to guarantee basic efficiency
    - A sequential-parallel race

BOP Correctness Protection

- Strategies
  - Classify data into different categories based on size, and patterns of accesses & values
  - Protect data in different granularity & strategies
  - Employ virtual memory paging support
- Correctness proved as an extension to data dependence theorems [similar proof as Allen & Kennedy, 2001]

BOP Correctness Protection

	Shared	**Checked**	**(Likely) private**
Properties to check	! (*lead*-write & *spec*-read)	*lead*: same value at BeginPPR & EndPPR	*spec*: write before read
Protection granularity	page/element	element	element

Overhead proportional to data size rather than accesses

Threads vs. Processes

- Threads: more efficient, fine-grained parallelism
- Processes: stronger isolation, coarse-grained parallelism

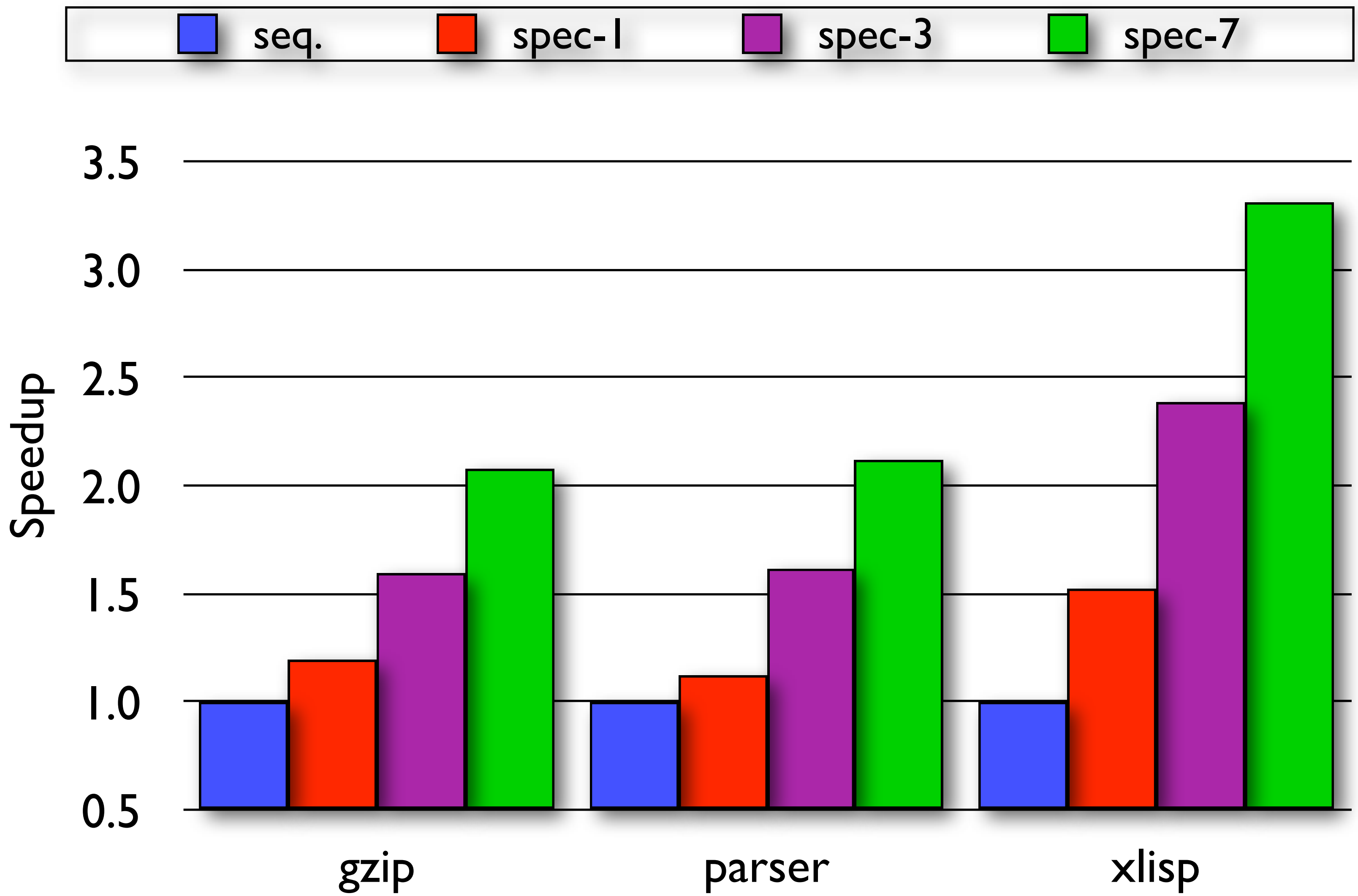
	Strong isolation	Weak isolation
Easy rollback	yes	no
Synchronization-free	yes	no
Relaxed dependence	yes (removes anti- & output-, allows value-based check)	no
Independent of hardware memory consistency	yes	no
Opportunistic parallelism	no	yes

Evaluation

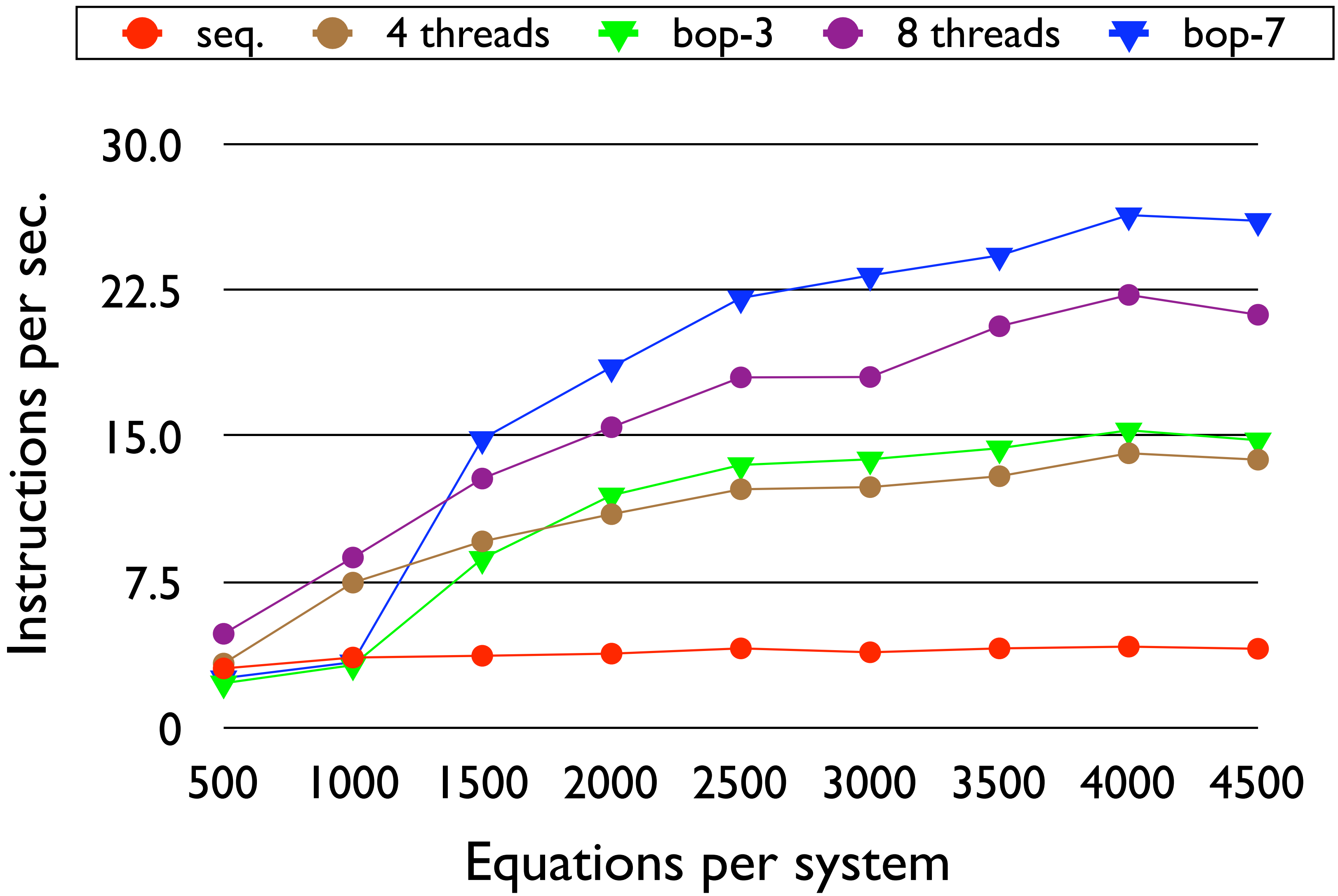
- Machine
  - Dell PowerEdge 6850 with 4 dual-core Intel 3.4GHz Xeon 7140M processors
- Compiler
  - GCC 4.0.1 with “-O3”
- Instrumentor
  - Modified GCC 4.0.1.

Gzip compressing an 84MB file

version	sequen-tial	speculation depth		
		1	3	7
times (sec)	8.46, 8.56, 8.50, 8.51 8.53, 8.48	7.29, 7.71 7.32, 7.47 5.70, 7.02	5.38, 5.49, 4.16, 5.71 5.33, 5.56	4.80, 4.47, 4.49, 3.10 2.88, 4.88
avg time	8.51	7.09	5.27	4.10
avg speedup	1.00	1.20	1.61	2.08


seq.
spec-1
spec-3
spec-7
3.5
3.0
2.5
2.0
1.5
1.0
0.5
Speedup
gzip
parser
xlisp

Intel MKL (Solving 8 Linear Systems)



Conclusions

- BOP exploits coarse-grained (input-dependent) parallelism
  - Strong isolation on process level with cost amortized by
    - granularity, pipelining & efficient checking
  - Basic efficiency (even in the worst case)
    - Overhead proportional to data size NOT to accesses
    - Understudy & minimal critical path
  - Programmability
    - Incremental parallelization based on partial information
    - No parallel programming or debugging
    - Only hints needed, no harm to correctness
    - PPRs allow error handling, GC, hidden dependences, and abnormal exits or entries

Thanks!

More details in “Software Behavior Oriented Parallelization”, PLDI’2007