Single-Chip Bottleneck

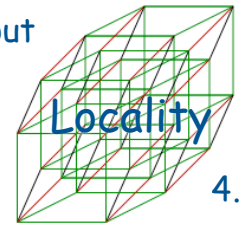
- Intel Core 2 Quad
  - four 2.40GHz cores
  - 8 64-bit flops per cycle per core
  - sustained memory bandwidth
    measured by STREAM benchmark
    - 5.3GB/s for four threads
    - 0.55 byte per cycle per core, 0.009 dword per flop
  - 8MB L2 cache
- Locality
  - How (in)frequent a program accesses main memory?
  - How much data does it actively use?
  - Must model long-range program behavior
- Program and machine balance
  - [Callahan, Cocke, Kennedy, JPDC 88] [Ding, Kennedy, JPDC 04]

reference affinity
[PLDI'04, ICS'05, POPL'06]

2. Data

whole-program locality [PLDI'03, PACT'03, LACSI'03, TOC'07]

1. Input



cache management, program tuning
[others at Ghent, Rice, and MTU]

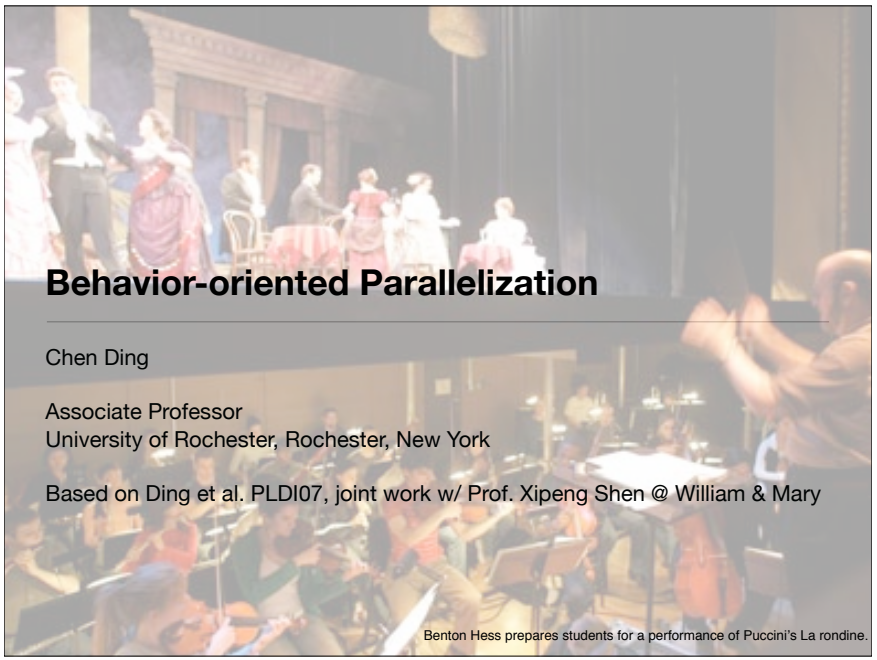
3. Code

4. Time

locality phases, active profiling
[ASPLOS'04, ExpCS'07, JPDC'07]

5. Environment

models of data, cache, and memory sharing [ISMM'06, PPOPP'08 poster, unpublished; others at MIT, NCSU, UWis, UMass/Amherst, etc]



Behavior-oriented Parallelization

Chen Ding

Associate Professor
University of Rochester, Rochester, New York

Based on Ding et al. PLDI07, joint work w/ Prof. Xipeng Shen @ William & Mary

Benton Hess prepares students for a performance of Puccini's La rondine.

High-level Parallelism

- High-level parallelism exists in many programs
  - E.g. utilities, interpreters, scientific computations
  - "[scientists] know how to write parallel algorithms"—Rudi Eigenmann
- To parallelize or paralyze

Complex code

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

Unpredictable parallelism

Example*:

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

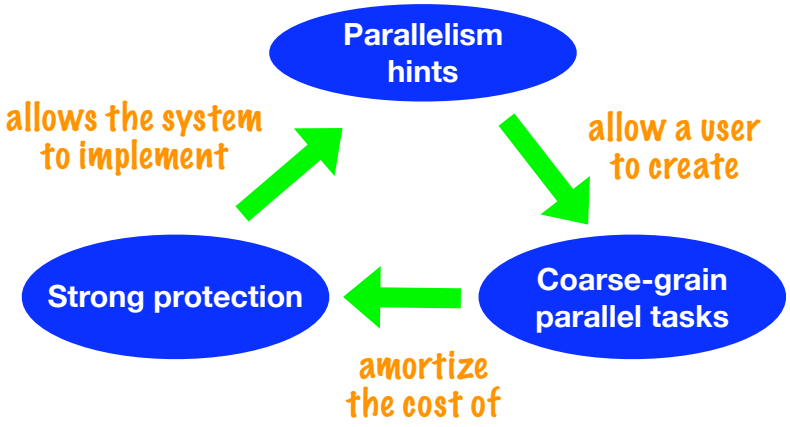
* Simplified Parser in SPEC2k by Sleator & Temperly

"we are interested in doing something now"—David Wood

Behavior Oriented Parallelization (BOP)

- Goal: parallelization using partial information
  - a user reading a small fraction of the source code
  - a profiler examining one or few inputs
- Approach: coarse-grain software speculation
  - speculate using processes (not threads)
    - protecting entire address space against unknown code
    - on-demand replication to remove all false dependencies
    - value-based checking to remove some flow dependencies
  - use granularity to hide overhead
  - mark likely parallelism to get course-grain tasks
    - possibly parallel regions (PPR)
    - affect performance but not correctness

The BOP Cycle



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

Basic semantics: at BeginPPR, fork a speculation process to execute from EndPPR.

Just hints of parallelism, no harm to correctness, unlike parallel sections, future, or transactional memory

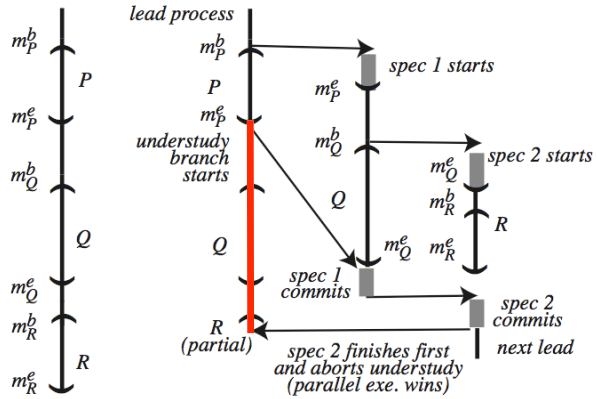
Process vs. Thread	Coarse-grain processes Strong isolation	Fine-grain threads Weak isolation
Opportunistic parallelism	no	yes
Free of false sharing		
Easy rollback	yes	no
Synchronization free		
Full data replication		
Independent of hardware memory consistency		
Value-based checking		
Run-time cost proportional to	data size	data access

"People who live in glass houses shall not throw stones."

BOP Correctness

- Conflict detection
  - protect data based on size, access & data values
    - value checking goes beyond dependence checking
    - page-level protection for global and heap data
      - methods for reducing false sharing
  - correctness proof [similar to Allen & Kennedy, 2001]
- Conflict resolution
  - feedback on the cause of conflicts
  - non-trivial program changes may be needed
    - changing sequential code only
    - no parallel programming or debugging

The Understudy Process



- The main overhead is off the critical path
- It is a race between sequential and parallel execution
  - "if you can't win, join them"

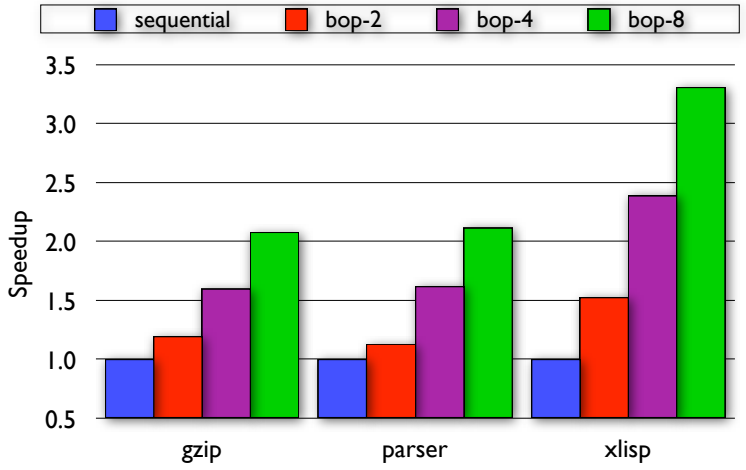
An Example of Value-based Checking

```
indents = 0;
while (...) // compile all functions
{
  BeginPPR(1);
  while (...) // compile next function
  {
     ... ...
     if ( "{" )  indents++;
     if ( "}" )  indents--;
  }
  EndPPR(1);
  ... ...
}
```


Gzip compressing an 84MB file

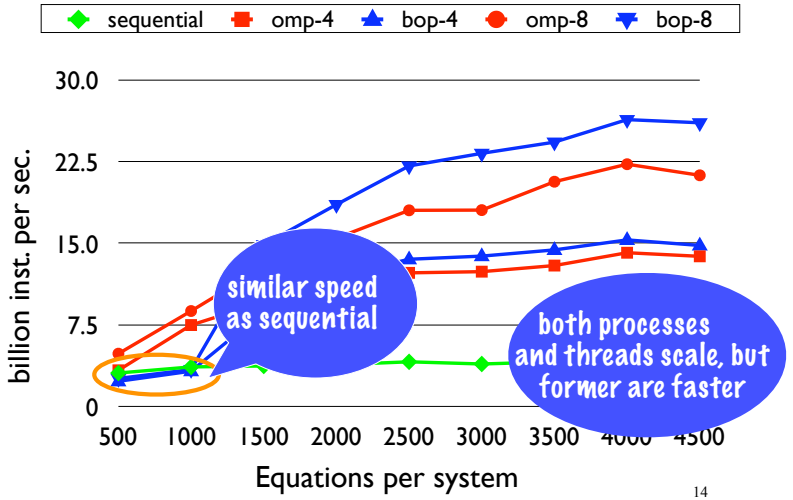
version	sequen-tial	speculation depth 1	3	7
times (sec)	8.46, 8.56, 8.50, 8.51 8.53, 8.48			
avg time	8.51			
avg speedup	1.00			

Dell PowerEdge 6850 with 4 dual-core Intel 3.4GHz, Xeon 7140M processors, GCC 4.0.1 with "-O3"



Dell PowerEdge 6850 with 4 dual-core Intel 3.4GHz, Xeon 7140M processors, GCC 4.0.1 with "-O3"

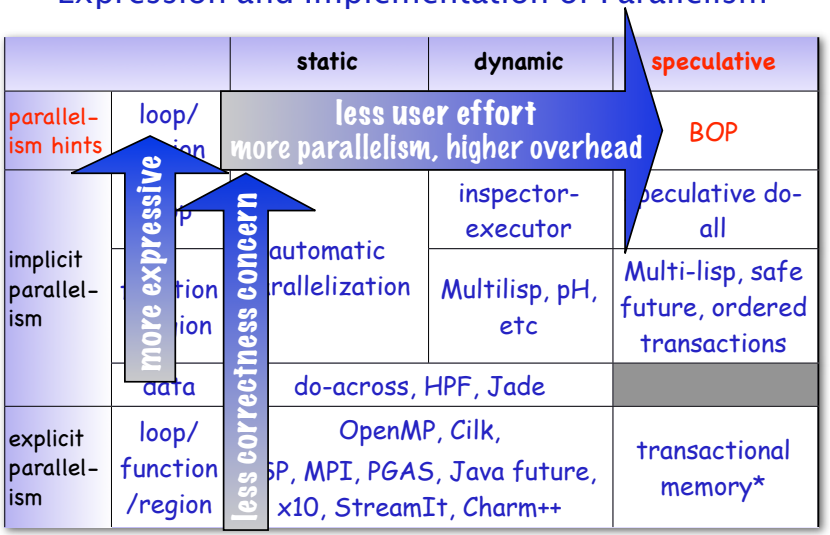
Intel MKL (Solving 8 Linear Systems)



Related Work (in Software)

- Loop based
  - speculative do-all (LPD) [Rauchwerger&Padua PLDI95]
  - guaranteed progress [Gupta&Nim SC98, Dang+TR02]
  - design space exploration [Cintra&Llanos TPDS05]
- Function or region based
  - safe Java future [Welc+ OOPSLA05], ordered transactions
- Many other related techniques
  - dynamic parallelization
    - inspector-executor, parallel functional languages
  - transactional memory [Wood yesterday]
  - thread-level speculation in hardware [Torrellas yesterday]
    - a limit study [Kejariwal et al. ICS06]
      - 12% max with infinite processors and zero overhead

Expression and Implementation of Parallelism



Summary of BOP Ingredients

- Strong isolation
  - complete, on-demand data replication
  - value- and dependence-based checking
- Run-time support
  - conflict detection, recovery, and tolerance
  - no worse perf. than sequential
- Programmability
  - only hints, no harm to correctness
  - no parallel programming or debugging
  - incremental parallelization
  - parallel execution despite hidden dependences