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 word 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]

Behavior-oriented Parallelization

Chen Ding

Associate Professor
University of Rochester, Rochester, New York

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

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

“we are interested in doing something now”—David Wood

reference affinity
- [PLDI’04, ICS’05, POPL’06]

1. Input
- Whole-program locality [PLDI’03, PACT’03, LACSI’03, TOC’07]

2. Data
- Cache management, program tuning
  - [others at Ghent, Rice, and MTU]

Locality

3. Code
- Locality phases, active profiling
  - [ASPLOS’04, ExpCS’07, JPDC’07]

4. Time
- Enables
  - Faster time to production code
  - Run off-the-shelf C programs
  - Lower operating cost

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- Models of data, cache, and memory sharing [ISMM’06, PPOPP’08 poster, unpublished; others at MIT, NCSU, UWis, UMass/Amherst, etc.]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 (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

Possibly Parallel Regions (PPRs)

```c
while (1) {
    get_work();
    ...  
    BeginPPR(1);
    step1();
    step2();
    BeginPPR(2);
    work(x);
    EndPPR(1);
    ...  
}
```

- 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

<table>
<thead>
<tr>
<th></th>
<th>Coarse-grain processes</th>
<th>Fine-grain threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opportunistic parallelism</td>
<td><strong>no</strong></td>
<td><strong>yes</strong></td>
</tr>
<tr>
<td>Free of false sharing</td>
<td><em>yes</em></td>
<td><em>no</em></td>
</tr>
<tr>
<td>Easy rollback</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synchronization free</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full data replication</td>
<td><em>yes</em></td>
<td><em>no</em></td>
</tr>
<tr>
<td>Independent of hardware memory consistency</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value-based checking</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Run-time cost proportional to</td>
<td><em>data size</em></td>
<td><em>data access</em></td>
</tr>
</tbody>
</table>

“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

An Example of Value-based Checking

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

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”

Gzip compressing an 84MB file

<table>
<thead>
<tr>
<th>version</th>
<th>sequential</th>
<th>speculation depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>times (sec)</td>
<td>8.46, 8.56, 8.50, 8.51, 8.53, 8.48</td>
<td></td>
</tr>
<tr>
<td>avg time</td>
<td>8.51</td>
<td></td>
</tr>
<tr>
<td>avg speedup</td>
<td>1.00</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>parallel-ism hints</th>
<th>static</th>
<th>dynamic</th>
<th>speculative</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop/region</td>
<td>less user effort</td>
<td>more parallelism, higher overhead</td>
<td>BOP</td>
</tr>
<tr>
<td>implicit</td>
<td>more expressive</td>
<td>automatic parallelization</td>
<td>inspector-executor</td>
</tr>
<tr>
<td>explicit</td>
<td>data do-across, HPF, Jade</td>
<td>OpenMP, Cilk, x10, StreamIt, Charm++</td>
<td>speculative do-all</td>
</tr>
<tr>
<td>function/region</td>
<td>less correctness concern</td>
<td>transactional memory*</td>
<td>Multi-lisp, safe future, ordered transactions</td>
</tr>
</tbody>
</table>
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