Revisiting the Sequential Programming Model for Multi-Core
Where we are?

- Multicore Era
- Lots of sequential code
- Hard to write parallel programs
Sequential Programming vs Parallel

- Intrinsic Limitation of sequential programming?
- No framework for automatic thread extraction
- A single legal program outcome
Parallelizing Compiler

- Most of the time in loops => TLP from loops
- Identify independency among dynamic instructions
- Non-Numerical Programs
How to Parallelize Loops (1)

**Decoupled Software Pipelining (DSWP)**

```c
while (ptr = ptr->next) {
    ptr->val = ptr->val + 1; // X
}
```

[Diagram of Decoupled Software Pipelining (DSWP)]

- Core 0
  - LD
  - X
  - DOACROSS
  - Iters * (LD Latency + Comm Latency)

- Core 1
  - LD
  - X
  - LD
  - Iters * LD Latency

- DSWP
  - LD
  - X
  - Iters * LD Latency
DSWP Speedup
How to Parallelize Loops (2)

Thread-Level Speculation (STAMPede)

- Alias Speculation
- Value Speculation
- Control Speculation

(a) A loop with ambiguous loop-carried data dependences.

(b) Speculatively parallel threads.
How to Parallelize Loops (3)

while (condition) {
   A: line = read();
   B: result = work(line);
   C: printf(result);
}

![Diagram showing parallelization of tasks A, B, and C across four cores.](image)
Example
Sequentail Programming

- Forces a single legal program outcome.
- What if i need one of many possible outcome.
Commutative

static int seed;

@Commutative
int Yacm_random() {
    int temp = seed / 127773L;
    seed = 16807L * (seed - temp * 127773L) - (temp * 2836L);
    if (seed < 0)
        seed += 2147483647L;
    return seed;
}
Another Example
Y-Branch

dict = start_dictionary();
while ((char = read(1)) != EOF) {
    profitable = compress(char, dict)

    @YBRANCH(probability=.00001)
    if (!profitable)
        dict = restart_dictionary(dict);
}
finish_dictionary(dict);
Hardware & Compiler Assumption

Misspeculation avoided if possible:

- Silent Store not causes a misspeculation
- Forward stored value to later threads

Avoid over-estimating dependences
## Parallelize Result

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>No. of threads</th>
<th>Speedup</th>
<th>Historic performance</th>
<th>Ratio**</th>
</tr>
</thead>
<tbody>
<tr>
<td>164.zip</td>
<td>32</td>
<td>29.91</td>
<td>5.38</td>
<td>5.56</td>
</tr>
<tr>
<td>175.vpr</td>
<td>15</td>
<td>3.59</td>
<td>3.71</td>
<td>0.97</td>
</tr>
<tr>
<td>176.gcc</td>
<td>16</td>
<td>5.06</td>
<td>3.84</td>
<td>1.32</td>
</tr>
<tr>
<td>181.mcf</td>
<td>32</td>
<td>2.84</td>
<td>5.38</td>
<td>0.53</td>
</tr>
<tr>
<td>186.crafty</td>
<td>32</td>
<td>25.18</td>
<td>5.38</td>
<td>4.68</td>
</tr>
<tr>
<td>197.parser</td>
<td>32</td>
<td>24.50</td>
<td>5.38</td>
<td>4.55</td>
</tr>
<tr>
<td>253.perlbmk</td>
<td>5</td>
<td>1.21</td>
<td>2.18</td>
<td>0.55</td>
</tr>
<tr>
<td>254.gap</td>
<td>10</td>
<td>1.94</td>
<td>3.05</td>
<td>0.64</td>
</tr>
<tr>
<td>255.vortex</td>
<td>32</td>
<td>4.92</td>
<td>5.38</td>
<td>0.91</td>
</tr>
<tr>
<td>256.bzip2</td>
<td>12</td>
<td>6.72</td>
<td>3.34</td>
<td>2.01</td>
</tr>
<tr>
<td>300.twolf</td>
<td>8</td>
<td>2.06</td>
<td>2.74</td>
<td>0.75</td>
</tr>
<tr>
<td>Geometric mean</td>
<td>17</td>
<td>5.54</td>
<td>3.97</td>
<td>1.39</td>
</tr>
<tr>
<td>Arithmetic mean</td>
<td>20</td>
<td>9.81</td>
<td>4.16</td>
<td>2.04</td>
</tr>
</tbody>
</table>

* Speedup needed to maintain existing performance trends, assuming 1.4× speedup per doubling of cores

** Ratio of actual speedup to expected historic speedup
Limitations

Manual intervention on the code

Misspeculations

Balance the pipeline?
How to Evolve

Compilers (OpenMP, Intel Compiler, Helix)

Transaction Memory MTX e SMTX

Heterogeneous Architectures

Parallel Programming Deterministic by Default

Parallel Language (Functional)
We need more power!

What about Power Consumption?
Thanks