Revisiting the Sequential Programming Model for Multi-Core

Matthew J. Bridges, Neil Vachharajani, Yun Zhang, Thomas Jablin, & David I. August

The Liberty Research Group
Princeton University
Source: Intel/Wikipedia
Intel Core2 Duo Die Photo: Source Intel
Parallel Programming Languages

Is easily Debugable, Maintainable, etc.?

Is Performance Retargetable?

Programmer Managed Speculation?

Parallelism Hard to Extract?

Legacy Application?
What prevents the automatic extraction of parallelism?
Lack of an Aggressive Compilation Framework
... prevented by Loop-Carried Dependences
An Aggressive Compilation Framework must parallelize inside of the loop body
An Aggressive Compilation Framework must speculate rare or predictable dependencies.
An Aggressive Compilation Framework must schedule dependences to reduce synchronization.
An **Aggressive Compilation Framework** must **be able** to optimize deep into the call tree.
An **Aggressive Compilation Framework** must be able to parallelize loops to efficiently utilize the available cores.
What prevents the automatic extraction of parallelism?

Lack of an Aggressive Compilation Framework

Sequential Programming Model
<table>
<thead>
<tr>
<th>Core 1</th>
<th>Core 2</th>
<th>Core 3</th>
<th>Core 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iter₁</td>
<td>Iter₂</td>
<td>Iter₃</td>
<td>Iter₄</td>
</tr>
</tbody>
</table>

**High Level View**

<table>
<thead>
<tr>
<th>Core 1</th>
<th>Core 2</th>
<th>Core 3</th>
<th>Core 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iter₁</td>
<td>Iter₂</td>
<td>Iter₃</td>
<td>Iter₄</td>
</tr>
</tbody>
</table>

**Low Level Reality**

<table>
<thead>
<tr>
<th>Core 1</th>
<th>Core 2</th>
<th>Core 3</th>
<th>Core 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iter₁</td>
<td>Iter₂</td>
<td>Iter₃</td>
<td>Iter₄</td>
</tr>
</tbody>
</table>
char *memory;

void * alloc(int size);  

void * alloc(int size) {
    void * ptr = memory;
    memory = memory + size;
    return ptr;
}

Can’t speculate the dependence
char *memory;

void *alloc(int size);

void *alloc(int size) {
    void *ptr = memory;
    memory = memory + size;
    return ptr;
}

**Can’t** speculate the dependence

**Can’t** schedule the dependence
char *memory;

@Commutative
void * alloc(int size);

void * alloc(int size) {
    void * ptr = memory;
    memory = memory + size;
    return ptr;
}

**Can’t** speculate the dependence
**Can’t** schedule the dependence
**Can** reorder the dependence
char *memory;

@Commutative
void * alloc(int size);

void * alloc(int size) {
    void * ptr = memory;
    memory = memory + size;
    return ptr;
}

Compiler does not preserve the existing sequential order, but does guarantee the existence of a sequential ordering.
What prevents the automatic extraction of parallelism?

Lack of an Aggressive Compilation Framework

Sequential Programming Model
The Liberty Research Group
http://liberty.princeton.edu