Of Programmers And Hardware: Transcending The Gap

Ulrich Drepper
Advertisement Claims

- What the CPU manufacturers want you to believe

Processor Performance

Time


MIPS

0 10000 20000 30000 40000 50000 60000

CPU MIPS
Advertisement Claims

- But most programs are not multi-threaded...
Advertisement Claims

- Ignoring SIMD performance

We have to work for this!
Parallelism

- Two types
  - Multi-core
  - Hyperthreads
- Not all cores in a package equally connected
- Planning concurrent execution
  - How much data has to be shared?
  - Functional units shared between cores/threads
CPU Structure

- Core
- FPU
- Core
- L3 Cache
- Memory Controller
- Core
- FPU
- Core
- L3 Cache
- Interconnect
System Structure
SIMD

- Single-Instruction/Multiple Data
  - Normal Arithmetic:
SIMD

- Single-Instruction/Multiple Data
  - Intel: MMX
**SIMD**

- Single-Instruction/Multiple Data
  - Intel: SSE

![Diagram showing SIMD operations with 128-bit data paths](image)
SIMD

- Single-Instruction/Multiple Data
  - Intel: AVX

```
  256
  △
  |    |
  v    v
  256  256
```

```
  256
  △
  |    |
  v    v
  256  256
```
Streaming Support

- Covers multiple data types, integer and floating-point
- Incomplete coverage
- Core:
  - Arithmetic
  - For video, audio, and photo processing
- Later extensions:
  - Comparisons
  - Logic operations
  - Vector operations
Non-Streaming Example

- Original code

```c
void levelscale(vec_float &dst, const vec_float &src)
{
    for (int i = 0; i < N; ++i)
        if (src[i] > 10)
            dst[i] = 10 + (src[i] - 10) * 9 / 10;
        else
            dst[i] = src[i];
}
```
Streaming Example

- Using SSE

```c
void levelscale(vec_float &dst, const vec_float &src)
{
    __m128 v10 = _mm_set_ps1(10.0f);
    __m128 v09 = _mm_set_ps1(0.9f);
    for (int i = 0; i < N / 4; ++i) {
        __m128 cmp = _mm_cmpgt_ps(src.f[i], v10);
        __m128 tmp = _mm_add_ps(v10,
                                _mm_mul_ps(_mm_sub_ps(src.f[i], v10), v09));
        dst.f[i] = _mm_blendv_ps(src.f[i], tmp, cmp);
    }
}
```
Utilize Processor completely

- Determine program factors
  - How much parallelism?
  - Which functional units?
  - Memory use:
    - Working set versus cache size
    - Memory bandwidth requirement
  - Synchronization requirements
Scheduling Decisions

- Two parties responsible:
  - Kernel:
    - Scheduling without insight into program
    - Optimal memory bandwidth
    - Cache sharing
    - Minimal energy use
  - Userlevel
    - Influence scheduling through affinity
    - Needs insight into CPU topology:
      - Connecting caches
      - Socket connections
## Core-Memory Gap

<table>
<thead>
<tr>
<th>Core</th>
<th>Array</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Core 3.7 GHz</td>
<td>233 MHz</td>
</tr>
<tr>
<td></td>
<td>to 3.7 GHz</td>
<td>266 MHz</td>
</tr>
<tr>
<td></td>
<td>Core 3.3 GHz</td>
<td>66 MHz</td>
</tr>
<tr>
<td></td>
<td>to 3.3 GHz</td>
<td>100 MHz</td>
</tr>
</tbody>
</table>
## Core-Memory Gap

<table>
<thead>
<tr>
<th>Core</th>
<th>Array</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>233 MHz</td>
<td>66 MHz</td>
<td>3.5:1</td>
</tr>
<tr>
<td>3.7 GHz</td>
<td>133 MHz to 266 MHz</td>
<td>14:1 to 28:1</td>
</tr>
<tr>
<td>3.3 GHz</td>
<td>100 MHz to 233 MHz</td>
<td>14:1 to 33:1</td>
</tr>
</tbody>
</table>
Memory Handling Decision

- **Kernel**
  - Implement memory policy

- **Userlevel**
  - Use cache lines efficiently
  - Use cache levels efficiently
  - Share caches where possible and useful
  - Prefetch cache lines
  - Use local memory
  - Needs insight into memory topology
  - Needs control over memory placement
Programming Language Effects

- For best performance:
  - Access the execution units (threads)
  - Access to kernel facilities (affinity)
  - Control over object placement
    - Cache line utilization
    - Alignment issues (cache associativity)
  - Fixed address space regions
    - For node binding
Language Spectrum

Little Control
Perl
Python

Much Control
Java
C
C++
Fortran
Programmer Progress

- OK to use scripting languages
  - Do not expect performance
- Automatic memory handling
  - Nice for fast programming
  - ... and safe programs
  - Does not allow memory optimizations
- Start with scripting
- Learn about hardware details
- Replace performance critical parts with C/C++
  - Help through OpenMP, streaming libraries, etc
- Proceed rewriting until performance goal is met
Questions?

drepper@redhat.com | people.redhat.com/drepper