Cilk for High Productivity Computing

Bradley C. Kuszmaul
(Presented by Charles E. Leiserson)
Supercomputing Technologies Research Group
MIT CSAIL
Cilk

A C language for dynamic multithreading with a provably good runtime system.

Platforms
- AMD Opteron
- Sun UltraSparc
- SGI Altix
- Intel Pentium

Applications
- virus shell assembly
- graphics rendering
- $n$-body simulation
- ★ Socrates and Cilkchess

Cilk automatically manages low-level aspects of parallel execution, including protocols, load balancing, and scheduling.
Example: Vector Addition

```c
void vadd (real *A, real *B, int L, int H){
    int i; for (i=L; i<H; i++) A[i]+=B[i];
}
```
Example: Vector Addition

To expose parallelism, convert loops to recursion.

Side benefit: Divide-and-conquer is good for caches!
Example: Vector Addition

C

```c
void vadd (real *A, real *B, int L, int H){
    int i; for (i=L; i<H; i++) A[i] += B[i];
}
```

Cilk

```cilk
void vadd (real *A, real *B, int L, int H) {
    if ((L+BASE) > H) {
        int i; for (i=L; i<H; i++) A[i] += B[i];
    } else {
        spawn vadd (A, B, L, (L+H)/2);
        spawn vadd (A, B, (L+H)/2, H);
        sync;
    }
}
```

Cilk is a **faithful** extension of C. A Cilk program’s **serial elision** is always a legal implementation of Cilk semantics. Cilk provides **no** new data types.
Example: Vector Addition

C

```c
void vadd (real *A, real *B, int L, int H) {
    int i; for (i=L; i<H; i++) A[i] += B[i];
}
```

Cilk

```c
void vadd (real *A, real *B, int L, int H) {
    if (L+BASE>H) {
        int i; for (i=L; i<H; i++) A[i] += B[i];
    } else {
        spawn vadd (A, B, L, (L+H)/2);
        spawn vadd (A, B, (L+H)/2, H);
        sync;
    }
}
```

Cilk is a **faithful** extension of C. A Cilk program’s **serial elision** is always a legal implementation of Cilk semantics. Cilk provides **no** new data types.
## Cilk Productivity

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>$T_1/T_{\text{serial}}$</th>
<th>$SLOC^*$ (Cilk)</th>
<th>$SLOC^*$ (MPI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>STREAM</td>
<td>1.062</td>
<td>85</td>
<td>658</td>
</tr>
<tr>
<td>PTRANS</td>
<td>1.004</td>
<td>87</td>
<td>2261</td>
</tr>
<tr>
<td>RandomAccess</td>
<td>1.002</td>
<td>161</td>
<td>1883</td>
</tr>
<tr>
<td>HPL</td>
<td>1.022</td>
<td>398</td>
<td>15608</td>
</tr>
<tr>
<td>DGEMM</td>
<td>1.015</td>
<td>373</td>
<td>?? †</td>
</tr>
<tr>
<td>FFTE</td>
<td>1.065</td>
<td>1085 ‡</td>
<td>1747</td>
</tr>
</tbody>
</table>

* “Source lines of code” omits comments and blank lines, but includes `.h` files (official count does not).

† MPI DGEMM uses the HPL parallel matrix multiplication. The framework is 184 SLOC.

‡ FFTW includes a Cilk interface (since it was a product of our research group). I wrote 76 SLOC for the framework.
## Speedups

<table>
<thead>
<tr>
<th>Platform</th>
<th>P</th>
<th>STREAM</th>
<th>PTRANS</th>
<th>RandomAccess</th>
<th>HPL</th>
<th>DGEMM</th>
<th>FFTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opteron 840</td>
<td>4</td>
<td>2.38</td>
<td>3.29</td>
<td>3.21</td>
<td>3.76</td>
<td>3.92</td>
<td>3.13</td>
</tr>
<tr>
<td>UltraSparc-III</td>
<td>16</td>
<td>11.25</td>
<td>11.32</td>
<td>8.78</td>
<td>14.55</td>
<td>15.16</td>
<td>14.67</td>
</tr>
<tr>
<td>UltraSparc-II</td>
<td>30</td>
<td>9.55</td>
<td>7.70</td>
<td>11.05</td>
<td>23.36</td>
<td>28.05</td>
<td>25.62</td>
</tr>
<tr>
<td>UltraSparc-IV</td>
<td>144</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>95.78</td>
</tr>
</tbody>
</table>

Many thanks to Sun Microsystems; the University of Rochester Department of Computer Science; and the MIT Department of Earth, Atmospheric, and Planetary Sciences for their donations of machine time to run these benchmarks.
Conclusion

• Cilk is *simple*, faithfully extending the legacy C language with only a handful of new keywords.
  ◦ *Cilk contains no new data types.*

• Cilk encourages *recursive* programming.
  ◦ *Divide-and-conquer exploits data locality for caches.*

• Cilk *scales down* to run on one processor with nearly the efficiency of C.
  ◦ *Fast C code ⇔ fast Cilk code.*

• Cilk *scales up* provably well, guaranteeing near-perfect linear speedup, assuming that
  ◦ *sufficient parallelism exists in the application, and*
  ◦ *the platform has adequate communication bandwidth.*
Cost of Programming

• Commodity codes are amortized over $10^4$ to $10^6$ more users than custom codes.

• Today’s custom scalable codes employ arcane programming models usable only by experts.

• Our research is focused on reinventing scalable computing as a seamless extension of commodity serial computing.
Current Research

- **JCilk**, a Java-based multithreaded language, fuses dynamic and persistent multithreading.
- **Adaptive thread and job scheduling** guarantees fair and efficient resource sharing.
- **Transactional memory** simplifies thread synchronization and improves performance compared with locking, especially for multicore processors.
- **Cilk-DXM** integrates Cilk with distributed transactional memory for clusters.
- **Parallel data-race detectors** can guarantee to find synchronization bugs efficiently.
- **Cache-oblivious algorithms** offer high performance for streaming file I/O through passive self-tuning.
World Wide Web

Cilk source code, programming examples, documentation, technical papers, tutorials, and up-to-date information can be found at:

http://supertech.csail.mit.edu/cilk

Download CILK Today!
HPC Challenge (Class 2)

*Most productivity:* Most “elegant” implementation of two or more of seven parallel benchmarks:

- **STREAM:** vector addition & scaling
- **PTRANS:** matrix transpose
- **RandomAccess:** eponymous
- **HPL:** PLU decomposition
- **DGEMM:** matrix multiplication
- **FFTE:** fast Fourier transform
- **b_eff:** bandwidth and efficiency