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

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;
  }
}
```

To expose parallelism, convert loops to recursion.

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

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];
}
```

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

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

~~Cilk~~ serial elision

```
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.

Cilk Productivity

Benchmark	T_1/T_{serial}	*SLOC* (Cilk)*	*SLOC* (MPI)*
STREAM	1.062	85	658
PTRANS	1.004	87	2261
RandomAccess	1.002	161	1883
HPL	1.022	398	15608
DGEMM	1.015	373	??†
FFTE	1.065	1085‡	1747

* "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

Platform	*P*	STREAM	PTRANS	RandomAccess	HPL	DGEMM	FFTE
Opteron 840	4	2.38	3.29	3.21	3.76	3.92	3.13
Altix 350	16	10.33	6.62	4.95	14.11	14.97	12.50
UltraSparc-III	16	11.25	11.32	8.78	14.55	15.16	14.67
UltraSparc-II	30	9.55	7.70	11.05	23.36	28.05	25.62
UltraSparc-IV	144					95.78	

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* $\Leftrightarrow$ *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