Cilk for High Productivity Computing

Bradley C. Kuszmaul
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

I implemented all 6 HPC Challenge benchmarks.
Distance to Desktop: # of Cilk keywords added to the serial program.

Benchmark	*SLOC* * *(Cilk)*	*SLOC* * *(MPI)*	*Distance to Desktop*
STREAM	58	658	11
PTRANS	81	2261	13
RandomAccess	123	1883	18
HPL	348	15608	41
DGEMM	97	?? †	19
FFTE	230	1747	35

* "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.

Performance

P	HPL Gflop/s	η	DGEMM Gflop/s	η	STREAM GB/s	η	PTRANS GB/s	η	FFTE Gflop/s	η
1	5.2		5.1		0.8		0.7		0.7	
2	9.4	89	9.7	96	0.9	56	0.5	36	0.9	67
4	17.3	85	19.7	97	1.8	57	0.9	33	1.8	68
8	30.8	73	35.7	88	2.9	46	1.7	30	2.9	55
16	52.5	63	64.9	80	4.0	32	3.3	29	4.0	38
32	88.6	52	118.9	73	6.8	27	6.1	27	6.8	32
64	101.6	30	248.0	76	14.0	28	11.6	26	14.0	33
128			463.1	71	25.0	25	18.3	20	25.9	31
256			943.0	73	44.2	22	27.2	15	49.5	29
384			1195.9	61	54.1	11				

What is limiting the speedup? The language or the hardware?

Performance vs. MPI

Cilk beats the best reported Altix numbers for PTRANS and FFTE.

	HPL		PTRANS		RandomAccess	FFTE	
P	*Gflop/s*	*η*	*GB/s*	*η*	*GUPS*	*GB/s*	*η*
Cilk32	**88.6**	**52%**	**6.1**	**27%**	**0.15**	**6.8**	**32%**
MPI32	**129.2**	**77%**	**2.6**	**11%**	**0.004**	**4.1**	**19%**
Cilk/MPI	**0.68**		**2.35**		**37.5**	**1.65**	
Cilk128			**18.3**	**20%**	**0.11**	**25.9**	**31%**
MPI128	**638.9**	**95%**	**7.5**	**8%**	**0.11**	**14.1**	**17%**
Cilk/MPI	**?**		**2.43**		**0.96**	**1.84**	

MPI performance taken from HPC web site for Altix 3700.

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

Acknowledgments

Many thanks to MIT Department of Earth, Atmospheric, and Planetary Sciences and NASA for their donations of machine time to run these benchmarks.

Keith Randall helped implement HPL in Cilk.