SC 05 HPCC Challenge Class II Awards

MTA-2

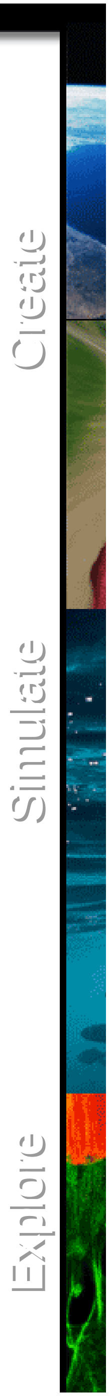
RA and FFT

Petr Konecny, Simon Kahan, and John Feo

Cray Inc

MTA 2

- Shared-memory, multithreaded architecture
  - Parallelism is used to tolerate latencies
  - Primary programming model is parallel loops
- No cache or local memory
- Zero-cost synchronization
- Compiler and runtime system responsible for implementing parallelism



Programming implications

- No code to optimize for cache
- No code to optimize for local memory
- No code to decompose data
- No code to communicate shared values
- No code to reserve, free, or manage hardware resources
- No code to synchronize hardware resources
- No code to assign/schedule software threads to hardware resources

Throw it all away ➔ short, concise programs

Source code is almost ALL math and science

RandomAccess

```
#define POLY 0x0000000000000007UL
#define NEXTRND(x) (((x) << 1) ^ ((int) (x) < 0 ? POLY : 0))

void random_access_update(int nupdates, int tableSize, unsigned Table[]) {
  unsigned bigloop = 1<<17;             // Number of outer loops
  unsigned bigstep = nupdates/bigloop;  // Number of updates in the inner loop

  assert((nupdates % bigloop)==0);

#pragma mta assert parallel
#pragma mta use 100 streams
  for (unsigned j = 0; j < bigloop; j++) {

    unsigned v = random_start(bigstep * j);

    for (unsigned i = 0; i < bigstep; i += 4) {
      v = NEXTRND(v);
      Table[v & (tableSize-1)] ^= v;
      v = NEXTRND(v);
      Table[v & (tableSize-1)] ^= v;
      v = NEXTRND(v);
      Table[v & (tableSize-1)] ^= v;
      v = NEXTRND(v);
      Table[v & (tableSize-1)] ^= v;
  } }
}
```

Kernel is only 14 lines

RA Statistics

Lines	Blank	Cmnts	NCSL	TPtoks
142	31	15	102	886

40 Processors
220MHz
160 GBytes shared memory

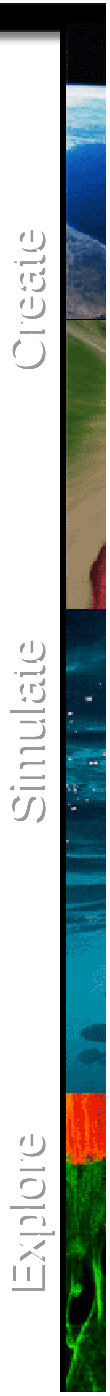
P	Gups	Sp
1	0.0493	1.0
5	0.2447	4.95
10	0.4899	9.95
20	0.9775	19.85
40	1.9450	39.49

Table size = 2^33 words

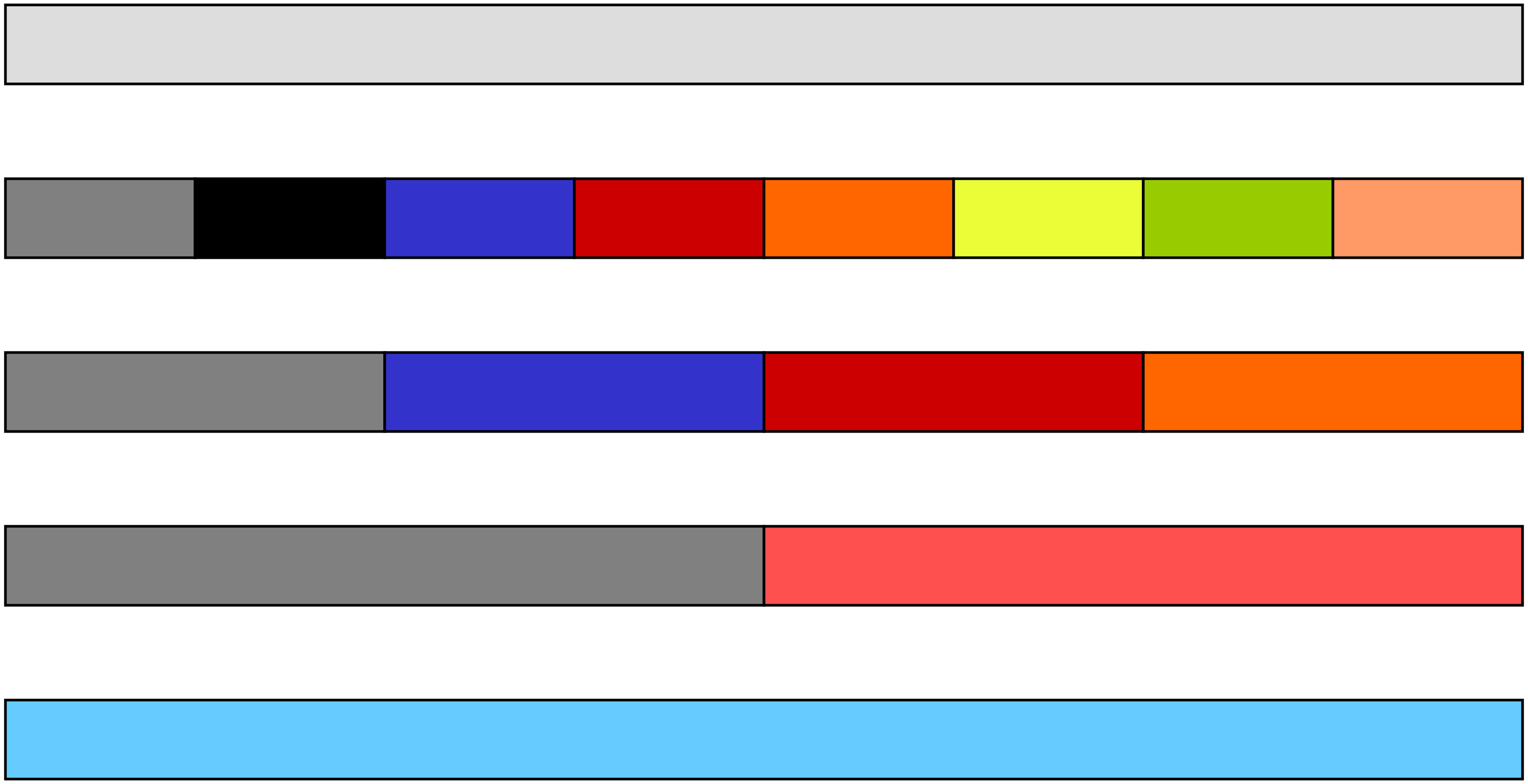
93% Utilization

0 Errors

4 cycles per update



FFT



Top level

```
void dfft(int n, int logn, double *a, double *w)
{ int i, l, j;
  double *v, *b, *c, *d;

  cft1st(n, a, w);

  i = 4; l = 8;

  for ( ; i <= logn / 2; i += 2, l *= 4) cftmd1(n, l, a, w);
  for ( ; i <= logn - 1; i += 2, l *= 4) cftmd2(n, l, a, w);

  cftlast(n, a, w);
}
```

Special routines for left most block
6 routines in all

ALERT – NO DATA TRANSPOSE

Butterfly

```
#pragma mta inline
void btrfly(j, wk1r, wk1i, wk2r, wk2i, wk3r, wk3i, a, b, c, d)
  int j;
  double wk1r, wk1i, wk2r, wk2i, wk3r, wk3i, *a, *b, *c, *d;
{ double x0r = a[j    ] + b[j    ];
  double x0i = a[j + 1] + b[j + 1];
  double x1r = a[j    ] - b[j    ];
  double x1i = a[j + 1] - b[j + 1];
  double x2r = c[j    ] + d[j    ];
  double x2i = c[j + 1] + d[j + 1];
  double x3r = c[j    ] - d[j    ];
  double x3i = c[j + 1] - d[j + 1];

  a[j    ] = x0r + x2r;
  a[j + 1] = x0i + x2i;
  x0r -= x2r;
  x0i -= x2i;
  c[j    ] = wk2r * x0r - wk2i * x0i;
  c[j + 1] = wk2r * x0i + wk2i * x0r;
  x0r = x1r - x3i;
  x0i = x1i + x3r;
  b[j    ] = wk1r * x0r - wk1i * x0i;
  b[j + 1] = wk1r * x0i + wk1i * x0r;
  x0r = x1r + x3i;
  x0i = x1i - x3r;
  d[j    ] = wk3r * x0r - wk3i * x0i;
  d[j + 1] = wk3r * x0i + wk3i * x0r;
}
```

Bit-reversal

```
double * bit_reverse(int n, double *w) {
  unsigned int i, mask, shift;
  double *v = new double[2 * n];

  mask  = 0x0102040810204080;
  shift = (int) (log(n) / log(2));

#pragma mta use 100 streams
#pragma mta assert no dependence
  for (i = 0; i < n; i++) {
      int ndx = MTA_BIT_MAT_OR(mask, MTA_BIT_MAT_OR(i, mask));
      ndx = MTA_ROTATE_LEFT(ndx, shift);
      v[2 * ndx]     = w[2 * i];
      v[2 * ndx + 1] = w[2 * i + 1];
  }

  free(w);
  return(v);
}
```

5 instructions per datum

FFT Statistics

Lines	Blank	Cmnts	NCSL	TPtoks
394	86	5	303	3250

40 Processors
220MHz
160 GBytes shared memory

P	GFlops	Sp
1	0.284	1.0
5	1.414	4.98
10	2.815	9.92
20	5.593	19.71
40	10.967	38.66

Data size = 2^30

96% Utilization

0.143 Relative error

1.3 flops per cycle