HPC Challenge 2006 Awards Competition: xlUPC on BlueGene/L

C. Caşcaval, G. Almási, C. Barton, E. Tiotto
G. Dózsa, M. Farreras, P. Luk, T. Spelce

IBM Research, IBM SWG Toronto, and LLNL

Environment

- **Benchmarks:**
  - HPL, FFT, Random Access and EP STREAM Triad
- **Software**
  - An experimental version of the IBM xlUPC compiler
  - An experimental version of the BG/L communication library
- **Blue Gene characteristics & installations**
  - BG nodes (2 procs. each) have 4M L3 cache, 512 MB local memory; connected by a 3D torus, 175 MB/s/link
  - Blue Gene/X – 1 rack, 2048 procs., 512 GB mem.
  - Blue Gene/W – 20 racks, 40K procs., 10 TB mem.
  - Blue Gene/L – 64 racks, 128K procs., 32 TB mem.

Global HPL

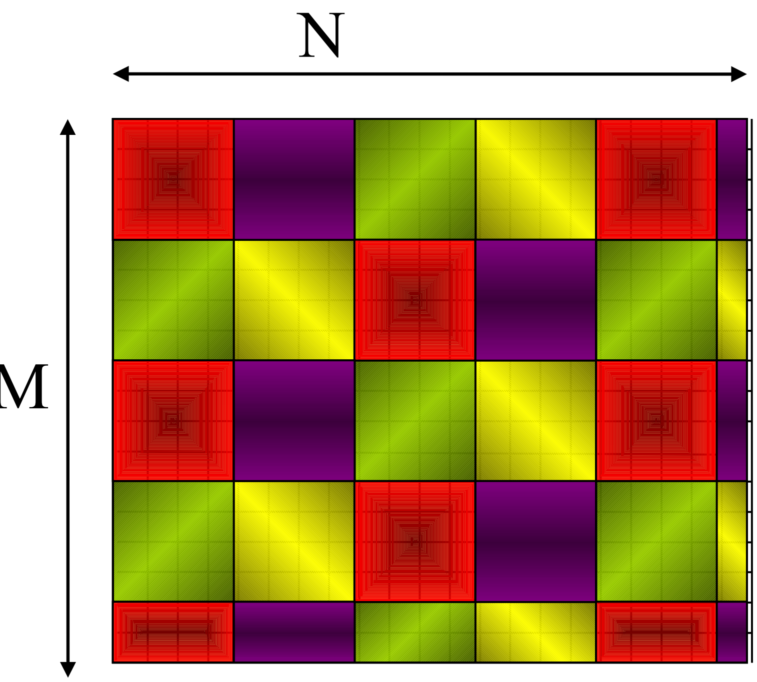
- **UPC naïve version – nice and simple code, low performance**
- **Optimizations:**
  - Calls to BLAS (ESSL in IBM speak) – we introduced multi-dimensional blocking of shared arrays
  - Collective communication – critically needed when scaling to thousands of processors
    - Added when UPC collectives supported (e.g., broadcast in backsolve)
    - update requires broadcast on subset of threads which is not supported in the UPC specification

Lines	Cmnts	NCSL	File
48	11	30	backsolve.upc
89	26	48	main.upc
52	12	35	matgen.upc
43	25	24	panel.upc
50	13	30	pivot.upc
45	16	23	swap.upc
45	24	15	tri_solve.upc
101	49	55	update.upc
63	22	28	hpl.h
536	198	288	Total

Multidimensional blocked data distribution in UPC

- **Syntax:** `shared [B][B] double A [M][N];`

new syntax



•Thread assigned to a[i][j]:

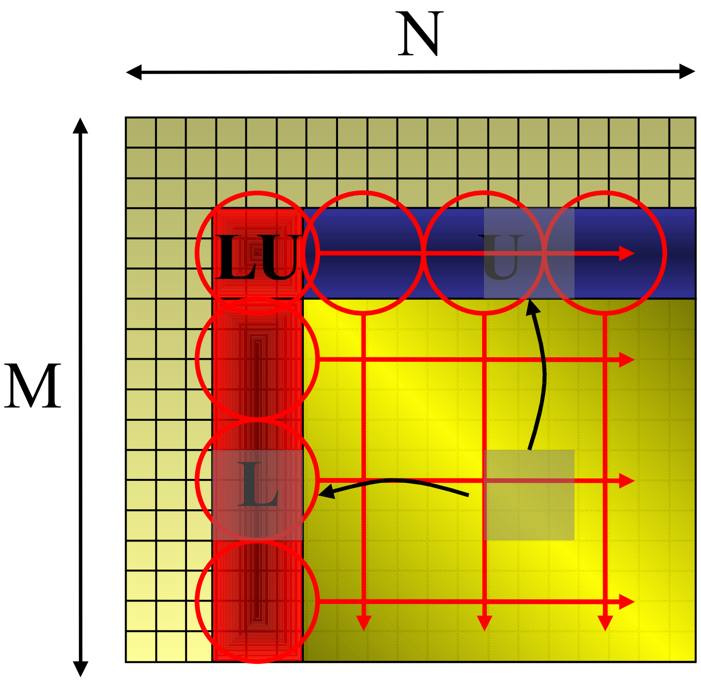
$$p(i,j)=\left(\left\lfloor\frac{i}{B}\right\rfloor\times\left\lceil\frac{N}{B}\right\rceil+\left\lfloor\frac{j}{B}\right\rfloor\right)\ mod\ \ THREADS$$

• Blocks are assigned sequentially, not in a checkerboard layout

Performance bottlenecks

- Comp/comm ratio low
  - **upc_memget()** calls overload the CPU that owns A[ii][k]
- Calls for collective communication (and subsets of threads)
  - No appropriate collective,
  - No communicators in UPC
- Collectives should also be used in: *backsolve, triangular_solve, outer_product, max_pivot*

```
void update(int k){
  upc_forall (ii; jj; ... &A[ii][jj]){
    upc_memget (a, &A[ii][k], B*B*8);
    upc_memget (b, &A[k][jj], B*B*8);
    c = (double *)&A[ii][jj];
    dgemm(..., a, b, c ...);
}}
```

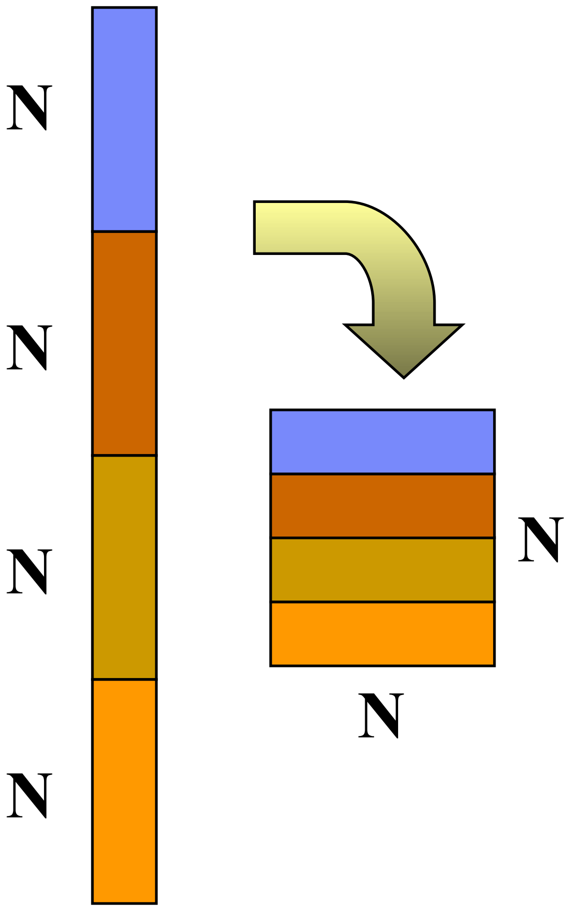


Performance

BlueGene Procs	Matrix Size	Gflops	Efficiency
1	5000	1.47	52.50%
64	44000	47.17	26.32%
256	85000	117.87	16.44%

- **Remaining issues**
  - Load balancing
  - Communication overhead (collectives)

Global FFT - Complex 1-D Discrete Fourier Transform (DFT)



Conventional algorithm: two-dimensional index mapping

- compute DFT of N columns
- multiply element (i,j) by W_{N*N}^{ij} (twiddle factors)
- compute DFT of N rows

DFTs can be done independently (in parallel)

- Matrix tranpose may be needed to make DFTs local
- FFTW library computes local DFTs

Global FFT – UPC code

Lines	Blank	Cmnts	NCSL	TP toks	
151	18	43	100	1018	fft.upc
59	14	23	22	160	fft.h
210	32	66	**122**	1168	**Total**
(121	*24*	*23*	*75*	*637*	*verify.upc)*

fftv1: `shared [N*N/THREADS] complex_t ComplexArray_t [N*N];`

fftv2: `shared [N/THREADS] complex_t ComplexArray_t [N*N];`

fftv1

transpose(X,A);
fftw_on_columns(A);
mult_by_twiddle(A);
transpose(A,Z);
fftw_on_rows(Z);
transpose(Z,A);

fftv2

local_copy_input(X,A);
fftw_on_columns(A);
mult_by_twiddle(A);
transpose(A,Z);
fftw_on_rows(Z);

Performance analysis

On 64 racks FFT performance is limited by the cost of transposes

Array size: **64** MBytes/thread

Data sent through cross-section each transpose: 32 MBytes/thread

cpubytes = 32 MBytes/η =
= **80 MBytes**
totalbytes = cpubytes * threads =
= **5120 GBytes**

Cross-section BW (64 x 32 x 32 torus)
2 wires/link x **32** x **32** x **2** links
Bandwidth =
4096 x 0.25 Bytes/cycle x 700MHz =
667 GBytes/s

fftv1 (3 transposes)

$T_{transpose}$ = totalbytes/BW = 7.68s
T_{fftw} = 1.8s;
$T_{twiddle}$ = 1.7s;
$T_{total} = 3 \cdot T_{transpose} + 2 \cdot T_{fftw} + T_{twiddle}$

$$\text{Performance} = \frac{5 \times n \times \log(n)}{T_{total}} \leq \mathbf{1843}\, \mathit{GFlops}$$

fftv2 (1 transpose)

$T_{transpose}$ = totalbytes/BW = 7.68s
T_{fftw} = 3.4s;
$T_{twiddle}$ = 2.2s;
$T_{total} = T_{transpose} + 2 \cdot T_{fftw} + T_{twiddle}$

$$\text{Performance} = \frac{5 \times n \times \log(n)}{T_{total}} \leq \mathbf{3131}\, \mathit{GFlops}$$

FFT Performance

BlueGene		Array	Arrays	fftv1	fftv2
Racks	Procs	Elements	TBytes	Gflops	GFlops
1	2048	2^32	0.13	51.29	54.29
4	8192	2^34	0.5	124.81	198.93
16	16384	2^36	2	512.70	742.90
64	65536	2^38	8	1115.00	N/A

Random Access

```
u64Int ran = starts(NUPDATE/THREADS * MYTHREAD);
upc_forall (i = 0; i < NUPDATE; i++; i) {
    ran = (ran << 1) ^ (((s64Int) ran < 0) ? POLY : 0);
    Table[ran & (TableSize-1)] ^= ran;
}
```

- **Each update is a packet**
  - Performance limited by latency, cross-section bandwidth
- **Compiler optimization:**
  - Identify remote update operations
- **Verification:** run the algorithm twice
- **Changes since last year:** optimized packet size
- **Lines of code:** 107

Theoretical GUPS limit on 64 rack BlueGene system

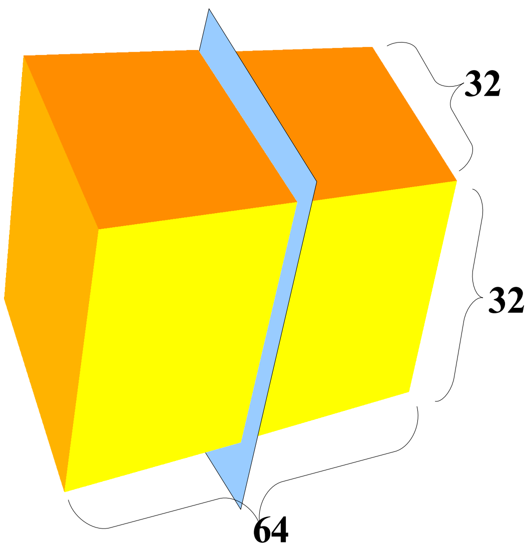
One packet per update (naïve algorithm!)

Update packets:

12 Byte header
4 Bytes opcode, type
4 Bytes target SVD
4 Bytes offset
8 Bytes update value
10 Bytes CRC + CF

42 Bytes on wire

$$\left\{\begin{array}{l}\left.\begin{array}{l}\textit{Packet size: 42 Bytes}\\ \textit{Wire speed: } 4\dfrac{cycles}{Byte}\end{array}\right\} \rightarrow 168\dfrac{cycles}{packet} \\ \textit{CPU speed: 700 MHz} = 1.4\dfrac{ns}{cycle}\end{array}\right\} \rightarrow \boldsymbol{P = 4.16\cdot10^6} \dfrac{packets}{second\cdot link}$$



Cross-section bandwidth:
64 x 32 x 32 torus:
2 wires/link x **32** x **32** x **2** (torus) = **4096** links
BW = $4096 \cdot 4.16 \cdot 10^6$ = $\mathbf{17 \cdot 10^9}$ packets/s

Half of all packets travel across the cross-section
Theoretical limit = **34 GUPS**

Random Access: Performance Results

BlueGene		Mem	GUPS	GUPS
Racks	Procs	TB	2005	2006
1	2048	0.25	0.56	0.58
2	4096	0.5	1.11	1.15
4	8192	1	1.70	2.28
8	16384	2	3.36	4.49
16	32768	4	6.10	8.83
32	65536	8	11.54	14.80
64	131072	16	16.72	28.30

Thank you!

Backup

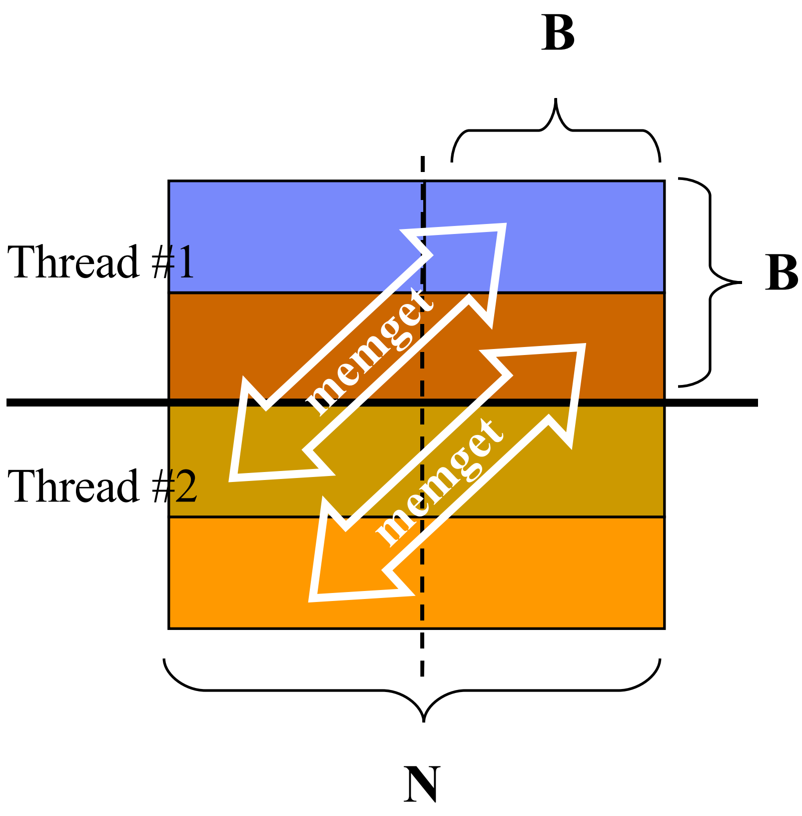
Global HPL Basics (Panel Factorization)

```
void parallel_panel () {
  for (k=...; k+=B) {
    panel (k, piv);
    swap_rows(k, piv);
    triangular_solve(k);
    update(k);
  }
}
```

```
void panel(col0, piv) {
  for (k = col0; ... ) {
    pivotRow = max_pivot(k);
    piv[k] = pivotRow;
    scale_column(k,pivotRow);
    swap_row(pivotRow);
    outer_product(k);
  }
}
```

- **Code:**
  - Follow **`dgetrf()`** floor plan
    - blocked factorization
  - Parallelize inner loops
    - blocks local to threads
  - Comm. granularity: **block**
- **Data:**
  - Need 2-D blocked distribution
    - Block locality, load balance
    - UPC syntax doesn't allow it!
    - ... so we extended UPC

FFT – Matrix transpose



All-to-all communication pattern

– bottleneck for Blue Gene

Blocked transpose

– blocksize B = N / THREADS

Each thread gets one B x B block from each other threads using upc_memgets

– no strided access with upc_memget

– we need B memgets for each block

Each block is tranposed in place at the destination

FFT – Matrix transpose: the code

```
upc_forall (i = 0; i < N; i += bsize; &B[ i*N ] )
   for (j = 0; j < N; j += bsize ) {
      // copy block to dest row by row
      complex_t * lb = (complex_t *)&B[i*N+j];
      for (unsigned k = 0; k < bsize; k++ )
         upc_memget( lb + k*N, &A[(j+k)*N + i], sizeof(complex_t) * bsize );
      // transpose block in place
      for (unsigned k = 0; k < bsize - 1; k++ )
       for (unsigned l = k + 1; l < bsize; l++ ) {
             complex_t c = lb[k*N+l];
             lb[k*N+l]   = lb[l*N+k];
             lb[l*N+k]   = c;
       }
    }
```

Transpose of A->B, shared arrays of N*N interpreted as (N, N) matrices

FFT – Multiplication by twiddle factors

- Z : shared array of N*N interpreted as (N, N) matrix
- multiplication of element (i,j) by W^{ij}_{N*N}, where $W^{ij}_{N*N} = e^{-2\pi*i*j/N*N}$

```
void multByTwiddleFactors(ComplexArray_t Z)
{
  for (ArrayIndex_t i = 0; i < N; i++ )
    upc_forall (ArrayIndex_t j = 0; j < N; j++; &Z[ i*N+j ] )
    {
      double x = ( 2 * M_PI * i * j ) / ( N * N );
      double tw_re =  cos(x), tw_im = -sin(x);
      Z[ i*N+j ].re = tw_re * Z[ i*N+j ].re - tw_im * Z[ i*N+j ].im;
      Z[ i*N+j ].im = tw_im * Z[ i*N+j ].re + tw_re * Z[ i*N+j ].im;
    }
}
```

EP Stream Triad

```
upc_forall (i = 0; i < VectorSize; i++; i) {
      a[i] = b[i] + alpha * c[i];
}
```

- **Embarrassingly parallel:** performance is gated by the individual node memory bandwidth
- **Important compiler optimization:**
  - Identify shared array accesses that have affinity to the accessing thread; transform them into local accesses
- **Verification:** random sampling
- **Lines of code: 90**

EP STREAM Triad – Performance Results

Processors	Problem Size	Memory Used	GB/s
2048	11,453,246,122	256 GB	1432.70
4096	22,906,492,245	500 GB	2865.35
8192	45,812,984,490	1 TB	5730.41
16384	91,625,968,981	2 TB	11460.65
32768	183,251,937,962	4 TB	22920.70
131072	733,007,751,850	16 TB	91627.49