IBM Research

2010 IBM HPC Challenge Class II Submission

George Almási **Barnaby Dalton** **Lawrence L Hu**
Franz Franchetti **Yaxun Liu** **Albert Sidelnik**
Thomas Spelce **Ilie Gabriel Tānase** **Ettore Tiotto**
Yevgen Voronenko **Xing Xue**

Our submission at a glance

- **2 ½ programming languages**
  - UPC and Coarray Fortran; Spiral for FFT
- **Two platforms**
  - Power clusters (UPC + CAF)
  - Blue Gene/P (UPC only)
- **One completely rewritten benchmark**
  - HPL with tiled array library (no UPC language extensions)
- **One new benchmark**
  - K-means clustering

Machines and compilers

http://www.alphaworks.ibm.com/tech/upccompiler

xlUPC

- Status: alpha
- UPC moving towards standardization

xlCAF

- Status: internal prototype
- Prioritized subsets in future Fortran releases

Spiral: academic project, supporting commercial company **http://spiralgen.com**

NCSA BluePrint cluster
32 nodes x 16 threads

IBM Poughkeepsie scaling cluster
32 nodes x 128 SMT threads

IBM TJ Watson Res. Ctr.
WatsonShaheen
4 racks x 4096 CPUs

Argonne Nat'l Labs
Surveyor + Intrepid
32 of 40 racks

Lawrence Livermore Nat'l Labs
Dawn
up to 16 racks

Benchmark descriptions 1/3: HPL, RA, Stream UPC and Coarray Fortran

- **CAF versions of stream, RA**

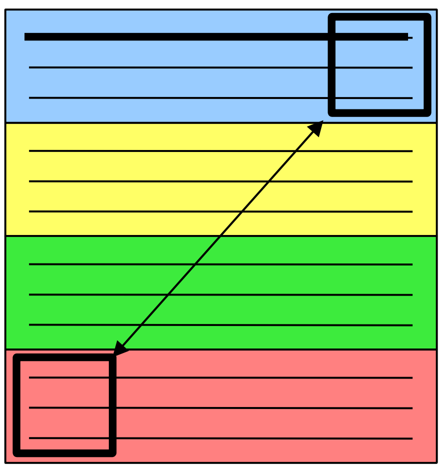
```
do i=1,updates_per_image
  a     = stream_next(a)
  image = iand(ishft(a, shift), mask)
  idx   = iand(a, index_mask)
  T(idx)[image+1]=ieor(T(idx)[image+1], a)
end do
```

```
subroutine stream_triad(a,b,c,alpha)
  real, intent(in) :: b(N)[*], c(N)[*]
  real, intent(out) :: a(N)[*]
  a(:) = b(:) + alpha * c(:)
end subroutine
```

- **Tiled array library (UPC and CAF)**
  - Designed for DLAs
  - Uses one-sided and collective communication primitives
  - UPC HPL implemented; CAF in progress

Benchmark descriptions 2/3: FFT UPC, CAF and Spiral

- **UPC, CAF: two variants of code**
- **Spiral: FFT in tensor language, generate code**



UPC and CAF: with Alltoall
- local DFT (contiguous)
- collect buffers into tiles
- global transpose (using Alltoall)
- local transpose tiles
- un-tile buffers
- twiddles etc.

UPC and CAF: naive
- local DFT (contiguous)
- line-by-line global transpose
- local transpose
- twiddles etc.

Spiral code:
- DFT scrambles data to tiled format, performs local transpose, twiddles
- global transpose (using Alltoall on contiguous data)
- Use MPI + OpenMP (although have experimented with UPC also)

Benchmark descriptions 3/3: k-means clustering

- **Problem: find few (K) representatives for large (N) set of points in (D)-dimensional Euclidian space**
  - Iterative method: K x N Euclidian distances/iteration
  - Important B/A kernel (e.g. SPSS)
  - Almost Embarrassingly Parallel (no scaling issues)
  - Simple code: 3 loops (N, K and D)
    - Bandwidth-gated
    - Intractable by today's compilers
    - Worthy successor to Stream

Trouble in k-means land

```
do n1=1,N
   mindist0 = 1.0e99
   kmin0 = -1
   do k1=1,K
      d0 = 0.0
      do d1=1,D
         diff0 = pointv(n1,d1)-clusterv(k1,d1)
         d0 = d0 + diff0*diff0
      end do
      if (d0 .lt. mindist0) then
         kmin0 = k1
         mindist0 = d0
      end if
   end do
   lnearestv(n1) = kmin0
end do
```

Low D count throws off unrolling

Odd number of FP – no FMAs

Branches throw off vectorizers

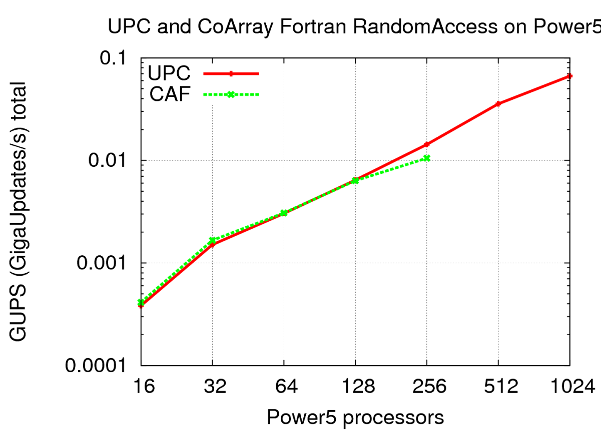
Best manual solution we found:

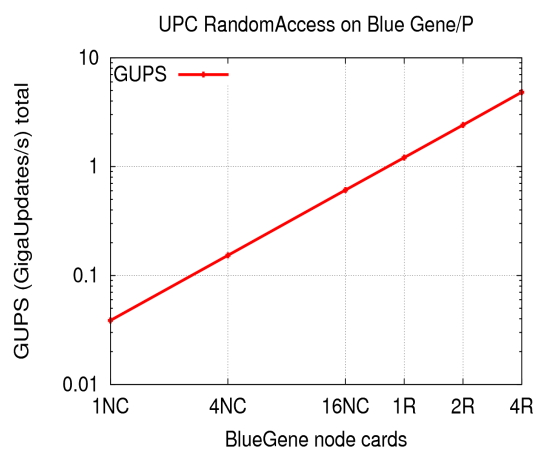
- Lay out data in non-intuitive way (K dimension first, D dimension second)
- Unroll + fuse K-loop into D-loop
- Manually deploy vector select() statements where available

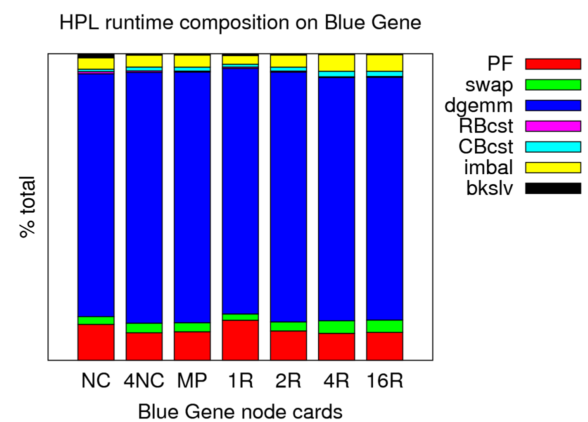
Performance summary 1/3: HPL, Stream, RA

HPL	Efficiency	ESSL eff.	PF	Load Imb	Comm
BG/P	51%	65%	10%	5%	8%
Power5	57%	72%	10%	4%	5%

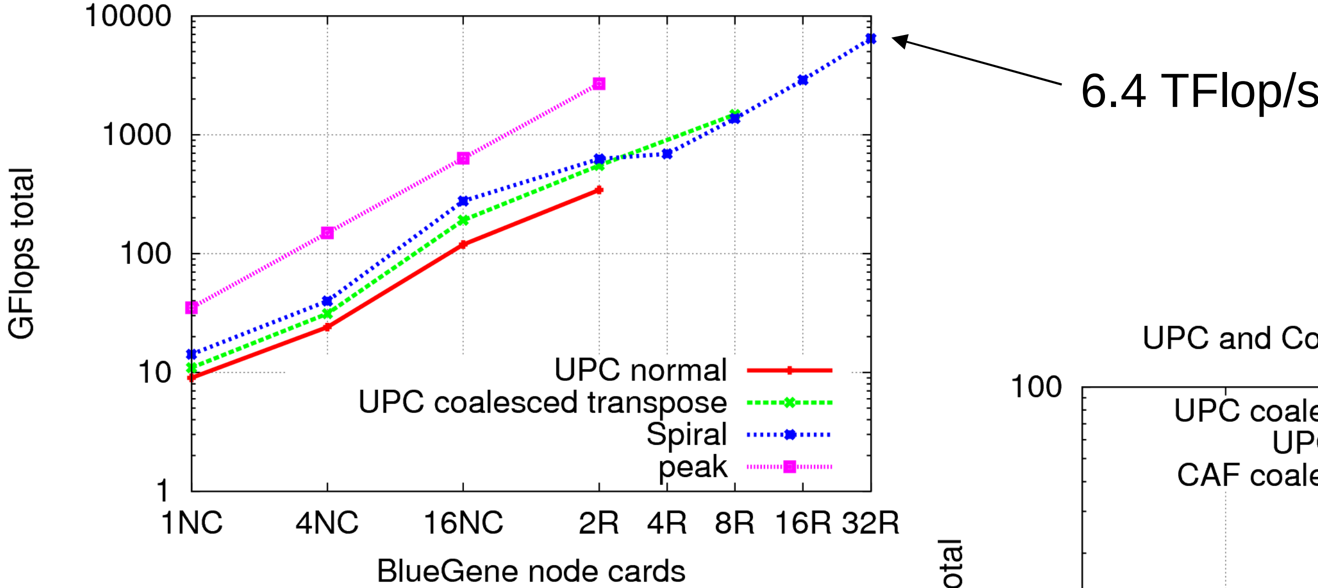
Stream	UPC naive	UPC opt	CAF
BG/P	32%	82%	N/A
Power7	50-60%		87%

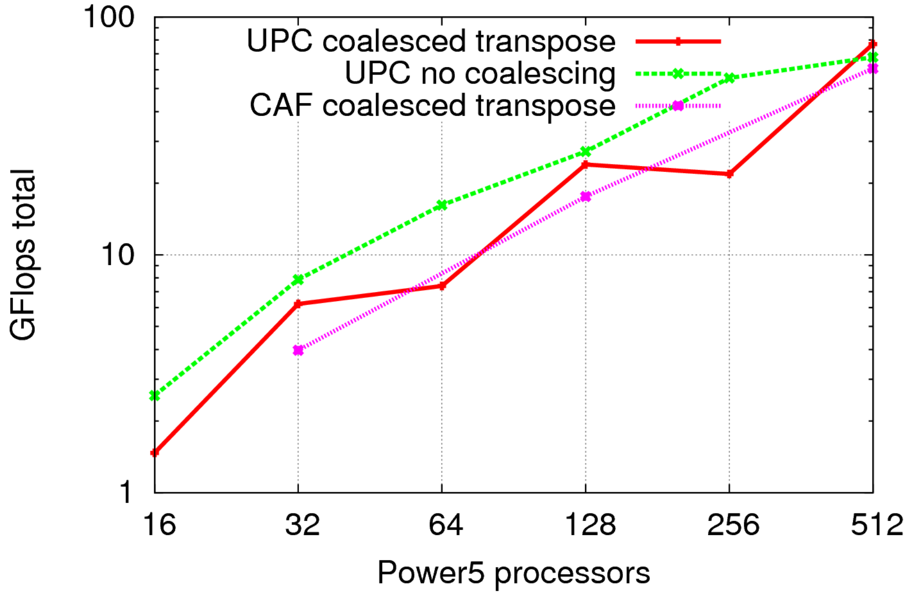




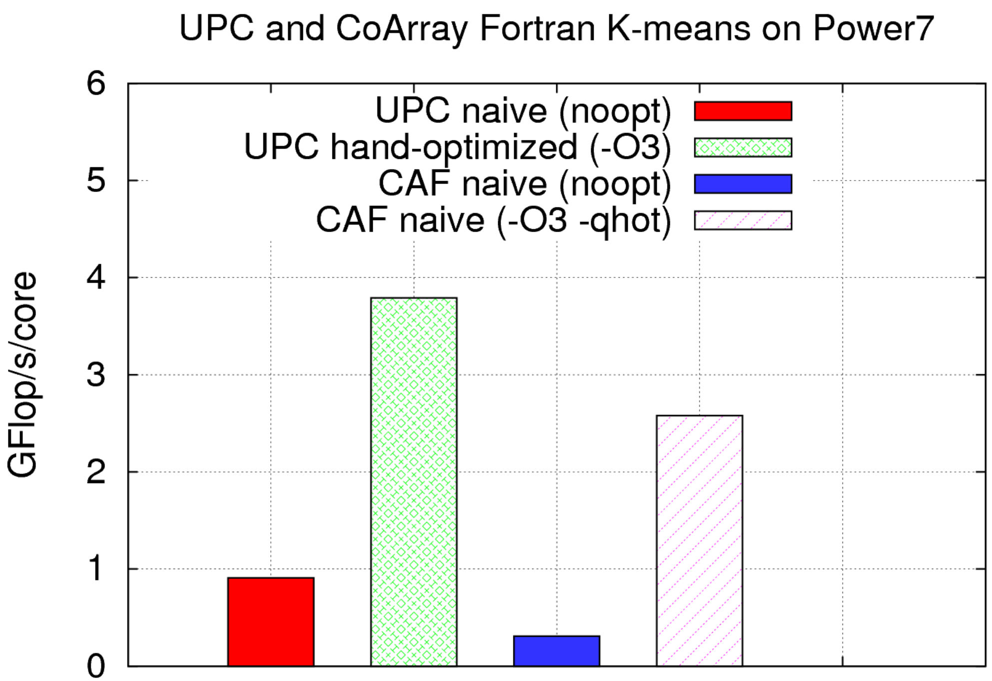


Performance summary 2/3: FFT





Performance summary 3/3: k-means



Peak perf. around 26 GFlop/s

On Blue Gene/P:

	GBytes/s
Peak BW	3.4
Stream BW	2.8
K-means naive	0.58
K-means hand-opt	1.22

Productivity and elegance

- **Spiral elegant and concise; generates impenetrable code**
- **Split-phase UPC memget caused a lot of trouble**
- **Coarray Fortran much less verbose than UPC**
  - UPC drawback: abuse of pointers to remote objects
  - Except for operators
- **UPC not quite equipped for writing libraries**
  - Need to fix
- **Lines of code summary:**

	HPL	FFT	RA	Stream	k-means
UPC	1012+595	369	165	186	~ 600
CAF	N/A+ 192	304	134	58	361

Conclusion

- **CoArray Fortran and UPC:**
  - Under development at IBM; teams work closely together
  - Emphasis on PGAS languages as library builders; interoperability with MPI a goal
  - Upcoming milestones for PERCS, Blue Waters
- **We know how to program for scalability**
  - But do not forget single/multicore performance
- **Spiral exploits domain knowledge**
  - Very concise expression (1 tensor equation)
  - Potential for absolute best performance on many architectures

Backup

Performance summary – Blue Gene/P (UPC/Spiral)

	HPL	RA	FFT	Spiral
procs	GFlop/s	GUP/s	GFlops/s	GFlops/s
1 NC	246	0.038	11	14.2
4 NC	1017	0.153	31	39.8
1 MP	4061	0.61	191	276
1 rack	8115	1.21		
2 racks	16214	2.41	550	625
4 racks	31626	4.82		691
8 racks			1479	1370
16 racks	112904			2890
Summary	52% flat	linear		

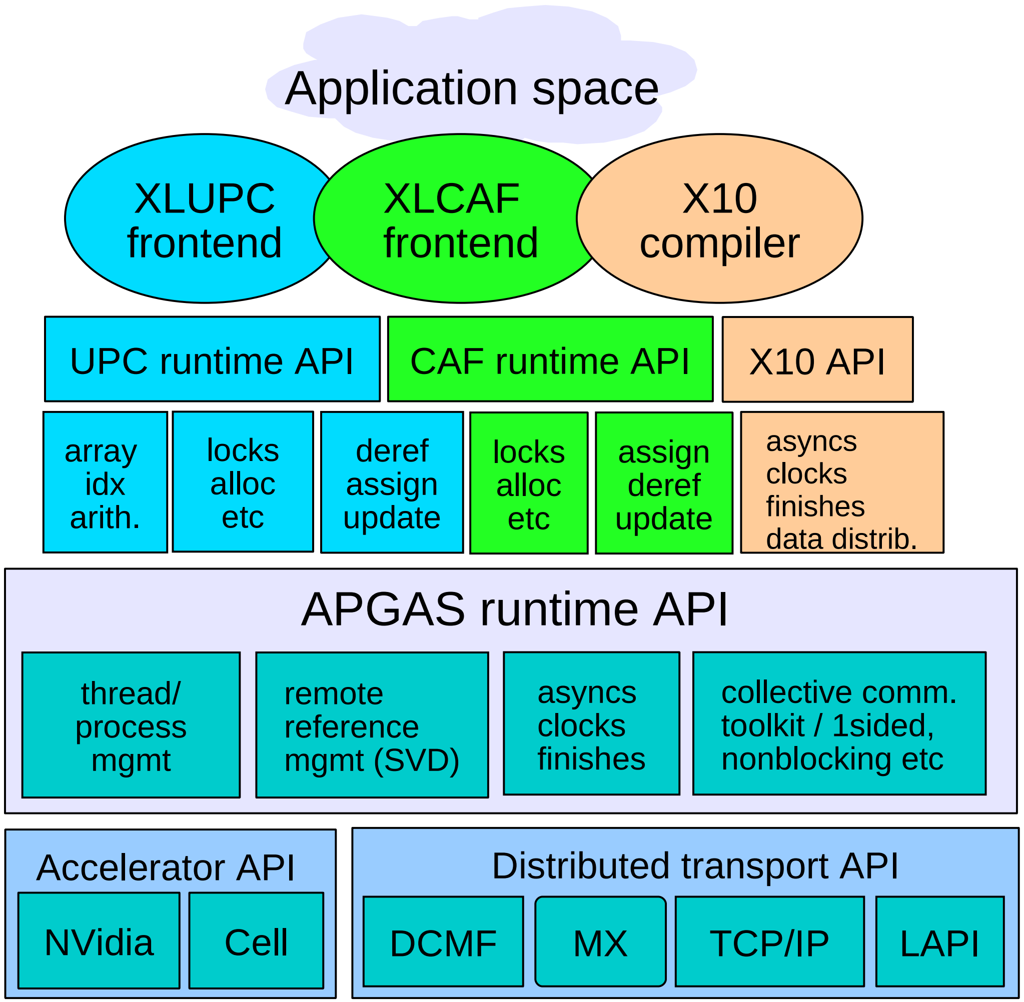
	Stream	k-means
procs	GBytes/s	GBytes/s
1	2.81	1.22
2	5.62	
4	12.07	
8	24.12	
16	48.30	
32	96.49	
64	193.09	
128	360.10	
	82%	

IBM TJ Watson Res. Ctr.
WatsonShaheen
All 4 racks

Argonne Nat'l Labs
Surveyor + Intrepid
up to 16 racks

Lawrence Livermore Nat'l Labs
Dawn
up to 16 racks

APGAS: one library to run them all



- **Support for UPC and CAF**
  - shared arrays; pointers-to-shared; locks; optimized collectives
- **Support for X10**
  - Asyncs & activities; remote references
- **Multiplatform**
  - Power, BG, Intel, Sun etc.
  - LAPI (IB, HPS), DCMF (BG), MX (Myrinet), TCP/IP sockets
- **Interoperable**
  - MPI

Bisection bandwidth calculation (Blue Gene/P)

# Nodes	Torus	Bisection (links)	BW (GB/s)	GUPS limit	FFT limit
32	4x4x2	32	13.6	0.32	39
128	4x8x4	128	55	1.30	176
1024	8x8x16	256	109	2.6	870
2048	8x16x16	512	217	5.2	1741
4096	16x16x16	1024	434	10.3	3482

Torus bisection = smallest diameter x 2 (torus) x 2 (half traffic)
Bisection Bandwidth = Bisection x 0.42 GB/s/link
GUPS limit = Bisection bandwidth / 42 bytes/packet
FFT Gflops = flops * BW / Bytes
FFT Gflops = 5* log(N) * N * N * Bandwidth / 3 * N * N * sizeof(cplx)