MPI performance "secrets"

George Almási

Introduction: the do's and don't's of BG/L MPI

- **Be mindful of memory**
  - Network is in userspace
    - Easily clobbered
    - All memory errors end up in communication library!
  - Memory is very limited
- **Be mindful of buffer ownership**
  - About to introduce more restrictions
- **Overlap computation & communication?**
  - Used to say: don't do it
  - Introducing two implementations to enable overlap
  - Performance largely untested

Summary

- **Communication libraries in System Software rel. 3 (Rochester)**
  - Interrupt driven operation
  - ARMCI/GA
- **Research Directions (Watson)**
  - Common external API infrastructure
  - UPC compiler & runtime

Interrupt driven communication

Charles Archer, Mike Blocksome, Brian Smith, Joe Ratterman, Pat McCarthy, Mike Mundy, Todd Inglett, Derek Lieber, Georghe Almasi, Jose Castanos, Jose Moreira, Jeff Parker

Interrupt driven communication: Summary

- **Traditional: network serviced by polling**
- **New: handle network device interrupts**
  - Torus “watermark” interrupt
    - Send & receive
  - VN mode Inter Processor Interrupts
- **Similar mechanism in VN and CP operating modes**
- **Better application response in certain situations**
- **But ... diminished overall performance**
  - Interrupt handling costs cycles!
  - “noise” in the system
  - cache pollution

Interrupt driven communication: Technical Challenges

- **No thread support in Compute Node Kernel**
  - Interrupt handling implies multiple execution contexts
- **HW: interrupts signal not trigger based**
  - Torus interrupt has to be disabled until handled
  - reenabled at the end
- **HW: watermark interrupts are critical**
  - Can be preempted by external input interrupts
- **BlueGene glibc not thread safe**
- **Network Hardware not thread safe**

Interrupt driven communication: Components of solution

- **New signals: SIGTORUS1, SIGTORUS2**
  - (names of these may change)
- **Recursive locks on glibc, network hardware**
- **System call: sc_torus_interrupt_ctl(action,mask)**
  - Action: enable/disable
  - Mask: bit vector of torus interrupt sources
  - Nested implementation (w/ counters)
- **Lock acquire/release: rts_torus_lock(), _unlock, _try**
- **Used by glibc and MPICH2**

Interrupt driven communication: Conclusion

- **MPI will start with interrupts disabled by default**
- **Interrupt behavior controlled by new environment variable (to be named)**
- **Performance compromise?**
  - Preliminary measurements indicate ~ 1000 cycles of interrupt handling overhead (0.7 µs)
  - Impact on cache and system noise to be evaluated

ARMCI/GA

Derek Lieber
Gheorghe Almasi
Jose Castanos
Sriram Krishnamoorthy

Brian Smith
Charles Archer
Joe Ratterman
Jose Moreira
Mike Blocksome
Mike Mundy
Pat McCarthy
Todd Inglett

ARMCI deployment

- **Done by IBM Rochester**
- **Port will reside at PNNL**
- **Will be deployed with Release 3**
- **Interrupts on by default**

- **Overall goals:**
  - Port ARMCI, GA
  - Port at least 1 GA application with community assistance
  - Don't break anything else
  - Show scaling to 1 rack

ARMCI/GA requirements

- **Messaging library API with one-sided operations**
  - Put, get, accumulate, wait, test, barrier, malloc, fence, lock/unlock, collectives
- **Peaceful co-existence with MPI**
- **Mechanism for overlapping computation with communication**

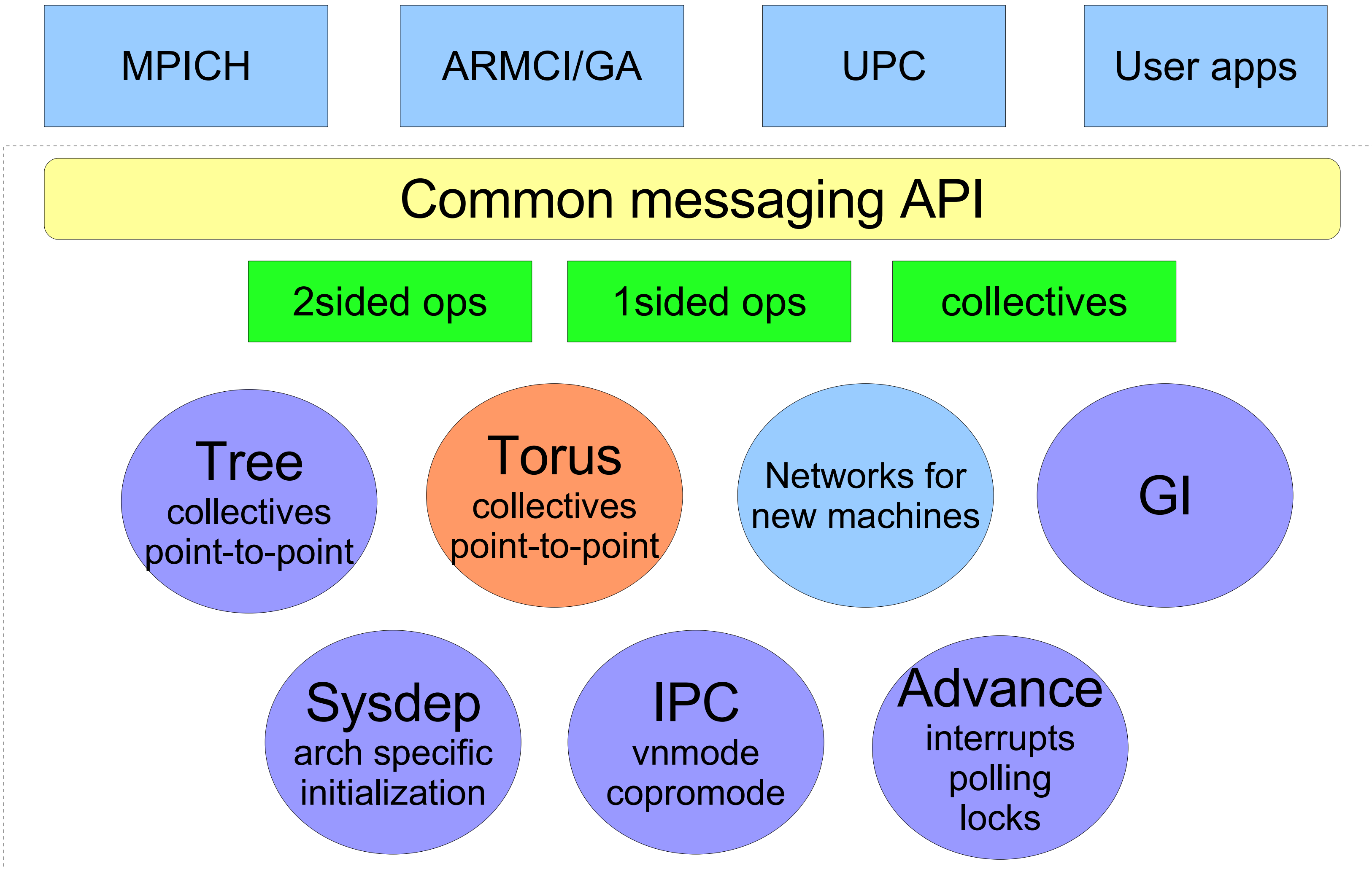
New at Research: Common Messaging API

Common Messaging API

- **One messaging interface ... to serve them all**
  - Like the One Ring, it's the stuff of legend and myth
  - ... maybe less sinister

- **Framework for ...**
  - Encapsulating algorithms already written
  - Allow new algorithms to be written easily
    - Think “toolkit” ... but that has legal implications as well
  - Allow portability (yes, we are thinking of /P)
  - Allow experimentation with new programming paradigms
  - A low(er) level of abstraction for messaging

Messaging infrastructure: BYOML



Common Messaging API: principles & components

- **Designed to be pollable**
- **Interrupt safe**
  - Thread safe
- **Non-blocking**
  - Can make blocking calls easily
- **Devices, methods & APIs**
- **Sysdep**
  - Mapping, initialization, configuration
- **2-sided point-to-point communication (MPI)**
- **1-sided point-to-point communication**
- **Collectives**
  - Coll. Net., Global Interrupts
  - Optimized torus collectives

Common Messaging API: Mapping & Initialization

```
// opaque datatype for holding singleton
typedef ...               BG_Messager_t;
typedef BG_Messager_t * BG_Messager_p;

// initialization, advance, query functions
void     BG_Messager_Init       (BG_Messager_p msgr);
unsigned BG_Messager_advance    (BG_Messager_p);
int      BG_Messager_mode       (BG_Messager_p);
unsigned BG_Messager_available  (BG_Messager_p);

// mapping
unsigned BG_Messager_rank       (BG_Messager_p);
unsigned BG_Messager_size       (BG_Messager_p);
int      BG_Messager_torus2rank (BG_Messager_p m, int, int, int, int);
int      BG_Messager_rank2torus (BG_Messager_p m, int rank,
                                  int *, int *, int *, int *);
```

Common API: 2-sided point-to-point messaging Types & Callbacks

```
Typedef ...             BG2S_t;
typedef BG2S_t        * BG2S_p;

// long message callback
typedef BG2S_t *(*cb_BG2S_Recv)    (const BGLQuad         * msginfo,
                                    unsigned                senderrank,
                                    const unsigned          sndlen,
                                    unsigned              * rcvlen,
                                    char                 ** rcvbuf,
                                    BG_Callback_t         * cb_info);
// short message callback
typedef void (*cb_BG2S_RecvShort) (const BGLQuad         * msginfo,
                                    const char            * sndbuf,
                                    const unsigned          sndlen);
```

Common API: 2-sided point-to-point messaging
Sending 2-sided messages

```
// 2-sided send
void  BG2S_Send   (BG2S_t                   * request,
                   const Callback_t         * cb_info,
                   BG_Messager_p              messager,
                   const BGLQuad            * msginfo,
                   const char               * sndbuf,
                   unsigned                   sndlen,
                   unsigned                   destrank);

// persistent send
void BG2S_Create (...);
void BG2S_Reset  (BG2S_t                    * request);
void BG2S_Start  (BG2S_t                    * sender);
```

Common Messaging API: Tree, GI collectives

```
void BGGI_Barrier       ();

void BGTree_Barrier     (int                pclass);

void BGTree_Bcast       (int                root,
                         void             * buffer,
                         int                nbytes,
                         int                pclass);

void BGTree_Allreduce   (const void       * sbuffer,
                         void             * rbuffer,
                         unsigned           count,
                         BGLML_Dt           dt,
                         BGLML_Op           op,
                         int                root,
                         unsigned           pclass);
```

Common Messaging API: One-sided messaging put, get, fences

```
void BG1S_Memput       (BG1S_p                  request,
                        const BG_Callback_t   * callback,
                        unsigned                destrank,
                        const char            * sndbuf,
                        unsigned                dstbase,
                        char                  * dstbuf,
                        unsigned                sndlen,
                        enum ...                consistency);

void  BG1S_Memget      (BG1S_t                * request,
                         ...,
                        bool                    isconsistent);

void  BG1S_Fence       (unsigned                destrank,
                        const BG_Callback_t   * callback);

void  BG1S_Allfence    ();
```

Common API: 1-sided consistency models ... and how to use them

- **Sequential consistency**
  - One outstanding op per rank
- **Relaxed consistency**
  - One outstanding PUT per peer
- **Location consistency (ala ARMCI)**
  - PUTs to same peer and overlapping addresses must be ordered
- **Zaphod's relaxed consistency**

- **UPC:**
  - sequential & relaxed consistency
- **ARMCI:**
  - Depends on whom you listen to
  - “Location consistency”
- **MPI one-sided**
  - Zaphod is your friend

UPC on BlueGene/L

C. Caşcaval, C. Barton, G. Almási,
Y. Zheng, M. Farreras, P. Luk, R. Mak
IBM Research and IBM SWG Toronto

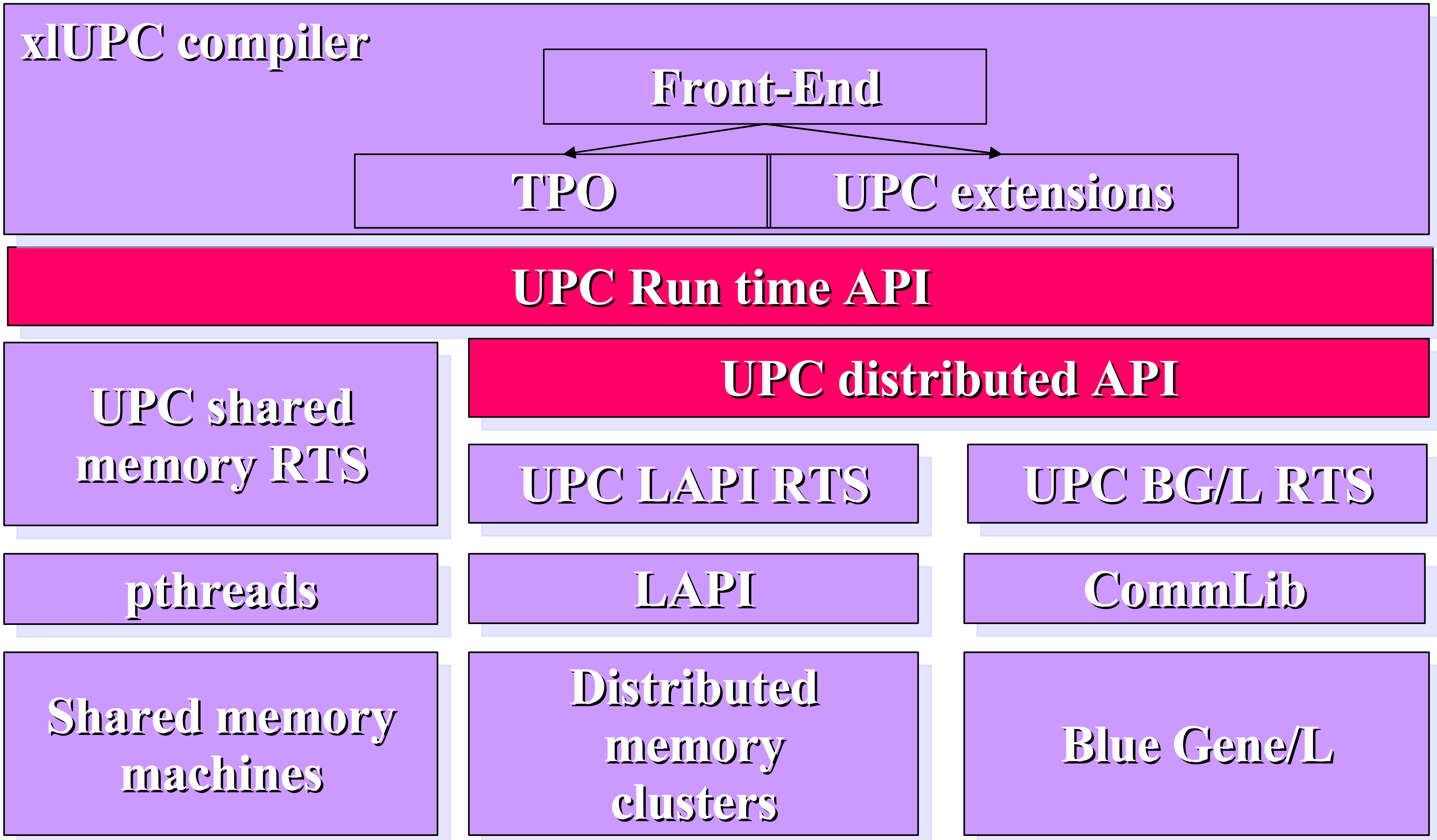
UPC On Blue Gene/L

UPC on BG/L: Overview

- **Shared memory programming model**
  - Partitioned Global Address Space (PGAS)
  - Shared or distributed memory
- **`shared` keyword**
  - Blocking factor
- **`upc_forall` loop**
  - With affinity test
- **UPC for AIX and Linux SMP: available on IBM alphaworks site**
- **Technology Preview, part of PERCS proposal**
- **Package extension for the IBM XL compiler v8.0**
- **2005 HCP Challenge Class 2 Award (shared)**

IBM XL UPC

UPC Compiler Architecture



Environment

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.

Software

- An experimental version of the IBM XL UPC compiler
- An experimental version of the BG/L communication library

Benchmarks:

- Random Access and EP STREAM Triad

GUPS Benchmark – Random Updates

```
shared u64Int Table[N];
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 is limited by network latency

Important compiler optimization:

- Identify update operations
- Translate them to one sided update in comm. library

Verification: run the algorithm twice

Lines of code: 111

GUPS: Performance Results

Processors	Problem Size 2^N	GUPS	Efficiency
1	22	0.00054	
2	22	0.00078	73%
64	27	0.02000	61%
2048	35	0.56000	51%
65536	40	11.54000	33%
131072	41	16.72500	23%

EP Stream Triad

```
shared double a[N], b[N], c[N];
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

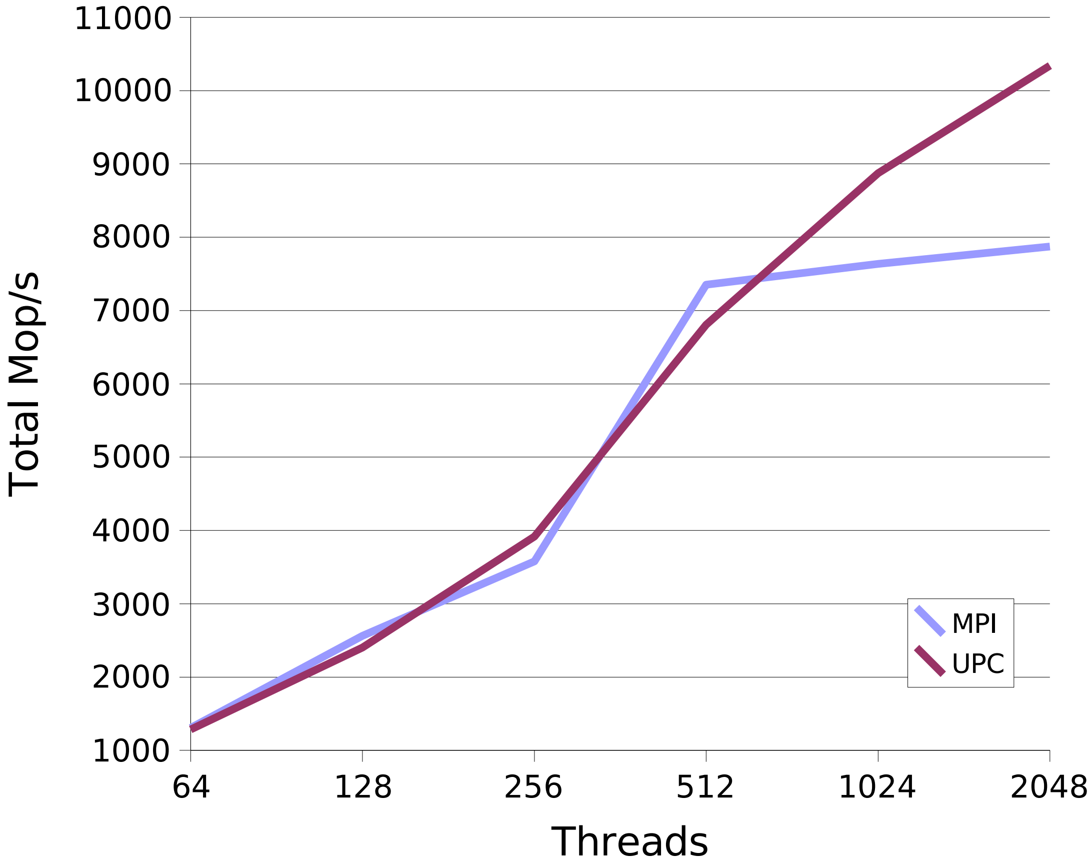
Verification: random sampling

Lines of code: 105

EP STREAM Triad – Performance Results

Processors	Problem Size	Memory Used	GB/s
1	2,000,001	45 MB	0.73
2	2,000,001	45 MB	1.46
64	357,913,941	8 GB	46.72
2048	11,453,246,122	256 GB	1472.00
65536	366,503,875,925	8 TB	47830.00
131072	733,007,751,850	16 TB	95660.00

NAS CG


11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
Total Mop/s
64
128
256
512
1024
2048
Threads
MPI
UPC

Discussion

We focused on the simplicity of code and on compiler and runtime optimizations, not on algorithmic changes

Most challenging issues:

- Overcome limitations in compiler indexing decisions and scaling the UPC runtime system to the max. machine size
  - How to index a 16 TByte array on a 32 bit machine?
- Obtaining single node performance comparable to C
  - Eliminate the shared memory translation overhead
- Reduce one-sided communication latency
  - Naïve UPC code tends to generate short messages

Acknowledgments

- Roch Archambault, Roland Koo, Raymond Luk (Toronto SWG)
- Jose Castanos, Siddhartha Chatterjee, John Gunnels, Manish Gupta, Fred Mintzer (Watson)
- Tom Spelce (LLNL)
- DARPA HPCS (financial support)

