Berkeley UPC

SESSION 3: Library Extensions

Dan Bonachea

Christian Bell, Wei Chen, Jason Duell, Paul Hargrove,
Parry Husbands, Costin Iancu, Rajesh Nishtala,
Mike Welcome, Kathy Yelick

U.C. Berkeley / LBNL

http://upc.lbl.gov

Berkeley UPC Library Extensions

- **Evaluating many experimental UPC extensions:**
  - Explicitly non-blocking memcpy library
  - Non-contiguous memcpy library
  - Point-to-point synchronization library
  - Value-based collectives
  - High-performance timers
  - Variable blocksize pointer addition
- **Availability:**
  - Everything in this talk is already implemented and available for evaluation in the 2.2 release

Value-based Collectives Library

- **Motivation:**
  - Improve the expressiveness of collectives interface
    - "UPC Collectives-for-dummies"
  - Important special case: scalar collectives
    - Helpful wrappers allow "one-liners" for scalar collectives
  - Especially useful in non-performance critical code
    - Initialization, throw-away test code, etc
- **Example:**

```
#include <bupc_collectivev.h>
double myvalue1 = ..., myvalue2 = ...;
double result1 = bupc_allv_reduce(double, myvalue1, 0, UPC_ADD);
double result2 = bupc_allv_broadcast(double, myvalue2, 0);
```

- **Implemented as a generic header file**
  - works with any UPC-1.2 compliant compiler
  - value-based versions of all the collectives except all_to_all

Value-based Collectives Library

- **Computational Collectives:**

```
#include <bupc_collectivev.h>
TYPE bupc_allv_reduce(TYPE, TYPE value, int rootthread, upc_op_t reductionop)
TYPE bupc_allv_reduce_all(TYPE, TYPE value, upc_op_t reductionop)
TYPE bupc_allv_prefix_reduce(TYPE, TYPE value, upc_op_t reductionop)

TYPE must be scalar
reductionop: UPC_ADD, UPC_MULT, UPC_AND, UPC_OR, UPC_XOR,
             UPC_LOGAND, UPC_LOGOR, UPC_MIN, UPC_MAX (no UPC_*FUNC)
```

- **Data Movement Collectives:**

```
#include <bupc_collectivev.h>
TYPE bupc_allv_broadcast(TYPE, TYPE value, int rootthread)
TYPE bupc_allv_scatter(TYPE, int rootthread, TYPE *rootsrcarray)
TYPE *bupc_allv_gather(TYPE, TYPE value, int rootthread, TYPE *rootdestarray)
TYPE *bupc_allv_gather_all(TYPE, TYPE value, TYPE *destarray)
TYPE bupc_allv_permute(TYPE, TYPE value, int tothreadid)

TYPE may be scalar or aggregate (struct or union) type
Array parameters are pointer-to-local for the calling thread
Ignored on non-root threads for rooted collectives
```

Misc Extensions: Timer library

- **Motivation: app tuning requires fast, accurate timers**
- **Provide standard interface to high-performance hardware wall-clock timers**
  - Lower overhead & more precise than POSIX gettimeofday()
  - Often by several orders of magnitude!
  - Analogous to MPI_Wtime(), but designed for lower overhead
- **Time is represented in integral "ticks"**
  - Abstract type **bupc_tick_t**
  - **bupc_ticks_now**() returns current tick count
    - Often expands to just a few instructions to read the CPU cycle counter
  - **bupc_ticks_to_us**() converts ticks to microseconds (64-bit int)
  - Query functions available to estimate the timer granularity & overhead
- **Example:**

```
bupc_tick_t start = bupc_ticks_now();
  compute_foo(); /* do something that needs to be timed */
bupc_tick_t end = bupc_ticks_now();
printf("Time was: %d microseconds\n", (int)bupc_ticks_to_us(end-start));
```

Misc Extensions: Timer library

gettimeofday / bupc timers (microseconds)	n2001 2.20GHz P4 Linux 2.4.21	seaborg 375MHz Powr3 AIX 5.2	phoenix Cray X1E	lemieux 1GHz Alpha EV6.8 Tru64 5.1
overhead (time to read the timer once)	1.006 0.054	2.058 0.121	105.891 0.208	0.245 0.602
granularity (min observable time betwn ticks)	1.000 0.040	1.000 0.103	66.000 0.150	976.000 0.600
conversion (time to convert val to microsecs)	0.010 0.101	0.003 2.016	0.990 -	- -

some OS's reportedly have gettimeofday() granularity ~10 millisec

Misc Extensions: Generalized Blocksize Pointer addition

- **Motivation:**
  - Allow arithmetic on (shared void *), when blocksize not compile-time constant
  - Especially useful to UPC library writers
    - because blocksize is often a dynamic input argument

shared void * **bupc_ptradd**(shared void *p, size_t blockelems,
size_t elemsz, ptrdiff_t elemincr);

- '**p**': the base pointer
- **'blockelems'**: the block size (number of elements in a block)
- **'elemsz'**: the element size (usually sizeof(*p))
- **'elemincr'**: the positive or negative offset from the base pointer

- **Example:**

```
bupc_ptradd(p, blockelems, sizeof(T), elemincr);
```

  - Returns a value q as if it had been computed:

```
shared [blockelems] T *q = p;
q += elemincr;
```

  - **Except blockelems need not be compile-time constant**