UPC Runtime Layer

Jason Duell
The Big Picture

The Runtime layer handles everything that is both:

1) Platform/Environment specific
   —So compiler can output one version of code for all platforms.

2) But also specific to UPC
   —So GASNet can remain language-independent.
Runtime Layer Laundry List

1) Shared pointer representation and manipulation
2) Pthread creation and management
3) Memory Management
4) Synchronization
5) Initialization code
Supported Runtime Environments

2 Main Axes:

1) Threads vs. Processes
2) Network vs. Shared Memory vs. both
   — Also; network vs. local synchronization mechanisms

We will support:

— Threads on a single SMP (all shared memory)
— Processes on a single SMP (all shared memory)
— Processes w/network (all network)
— Threads w/network (use both)

We won’t support (at least for now)

— Processes on SMP & network (using both)
   — Will only use network communications
Implementation goals

**Speed:** compile time resolution instead of run-time checks wherever possible.

**Parsable by compiler** (for compilers that generate straight to assembly)
   —Inline functions instead of macros where possible.

**Clean, maintainable implementation**
   —But have it done yesterday
Shared Pointer Representation

```c
struct naïve_shared_ptr {
    void * addr;
    uint thread;
    uint phase;
};
```

- Provide phaseless shared pointer type (for both phaseless and default cyclic).
  — Can omit phase field.
  — If pure shared memory, this can just be a pointer

- 64 bit platforms: may be able to stuff some fields into unused top bits of pointer.

- Using offset instead of address may save space
  — But might make casts to local slower…

- Solution: provide abstract type and operations.
Thread-specific data

All unshared global & static declarations must be have a copy per pthread.

Solution #1: Put all variables in a big struct, and make 1 copy of it per thread.
   —Need to effectively eliminate separate compilation (slow).
   —Data no longer initialized by linker
   —Object files not readable by nm, etc.

Solution #2: Put all variables in single link section—make 1 copy of section per thread, and use pointer & offset into section to reference variables.
   —Solves initialization, separate compilation, object format.
   —But involves nonstandard compiler and linker directives.
Heap Management

GASNet provides a single, fixed network-accessible shared memory region to the Runtime.

The Runtime must divide it among threads, and manage separate local and global heap for each thread.

Also must prevent regular C heap from expanding into shared region: hook malloc/free to our own, bounded heap.
Shared Memory Allocation

- Memory allocator
- "Handle-safe" lock

Diagram showing memory allocation across multiple nodes.
Synchronization

Pure Shared Memory environments:
—Runtime provides synchronization via pthreads or System V IPC mechanisms.

Networked environments:
—GASNet provides synchronization across processes via the network
—Runtime provides it between threads in the same process.
Allocating/Initializing Shared Data

Initialization of shared data can be tricky:

```c
extern shared int array[THREADS];
shared int *p = &array[8];
```

Thread-specific data: can no longer trust linker to initialize addresses for unshared global/static pointers:

```c
int foo;
int *pfoo = &foo;
```

Solution: per file initialization functions to handle complex cases

—Must be able to run in arbitrary order
—Runtime may provide helper functions for compiler.
Implementation Plan

1) Processes with shared memory:
   —In progress: should be done by mid-June.
   —Allow compiler correctness testing and optimization work to proceed.

2) Processes with network
   —Less than a month additional effort.
   —Allow GASNet testing, and ports to multiple networks.

3) Threads support
   —Trickiest implementation issues.

4) Ports to other platforms trivial given a GASNet implementation for the network.
   —Mainly compiler/linker-specific hooks for TSD.