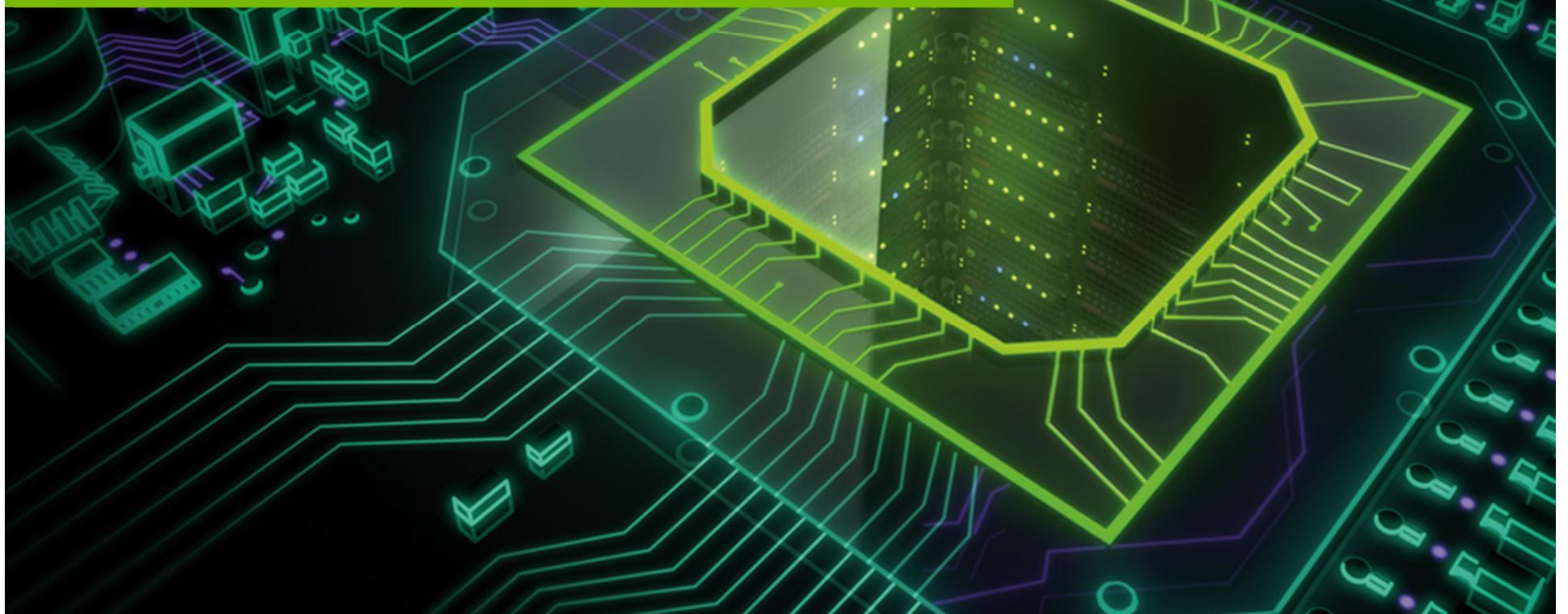


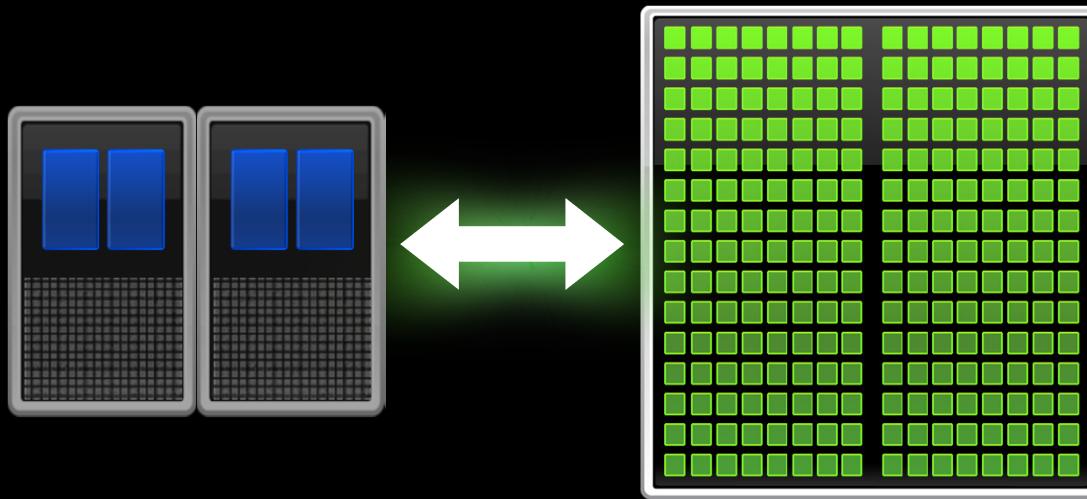
Designing a Unified Programming Model for Heterogeneous Machines

Michael Garland

Programming Systems & Applications
NVIDIA Research



Heterogeneous Parallel Computing



Multicore CPU

Fast Serial Processing

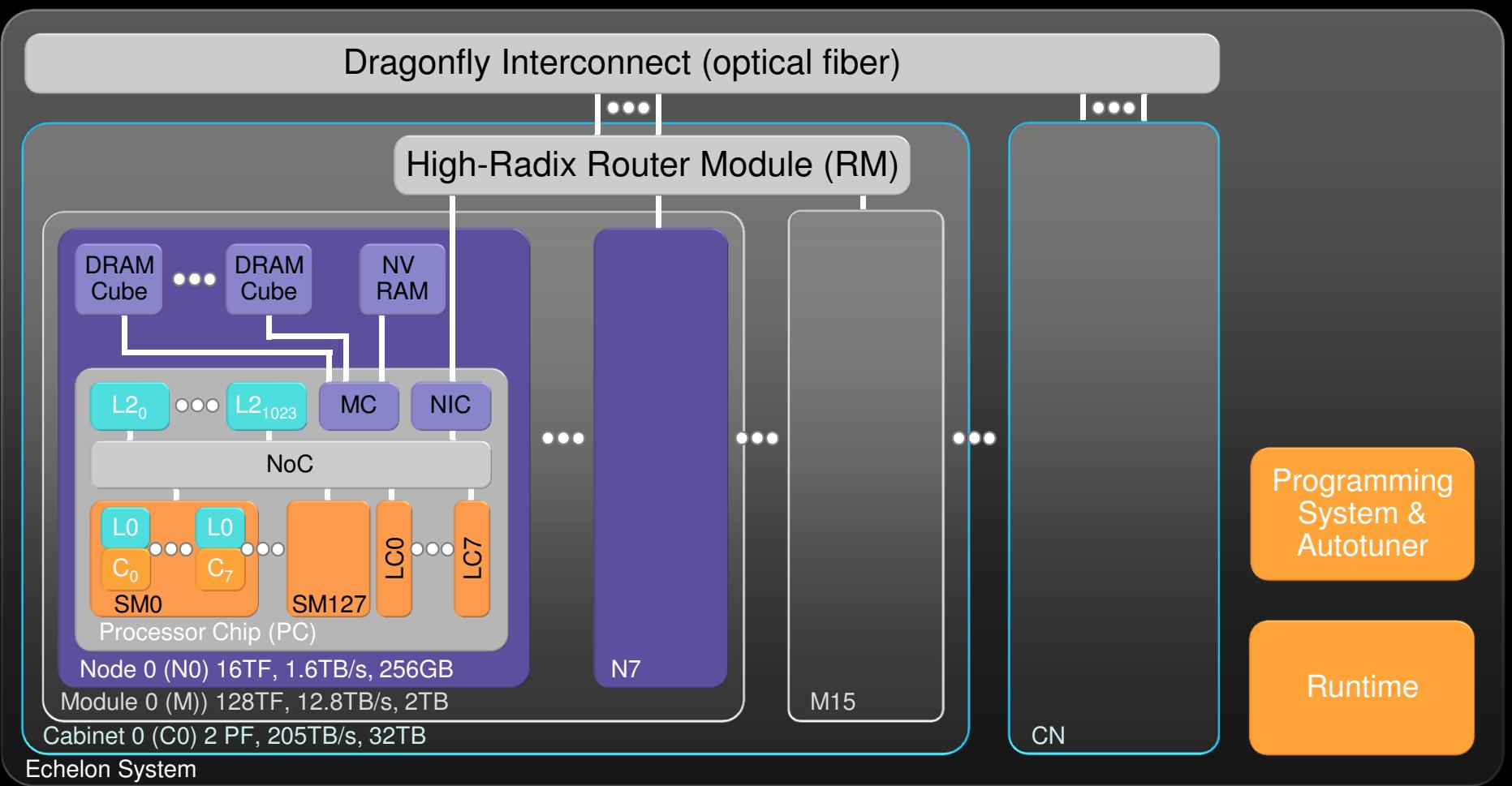
Latency-optimized
cores

Manycore GPU

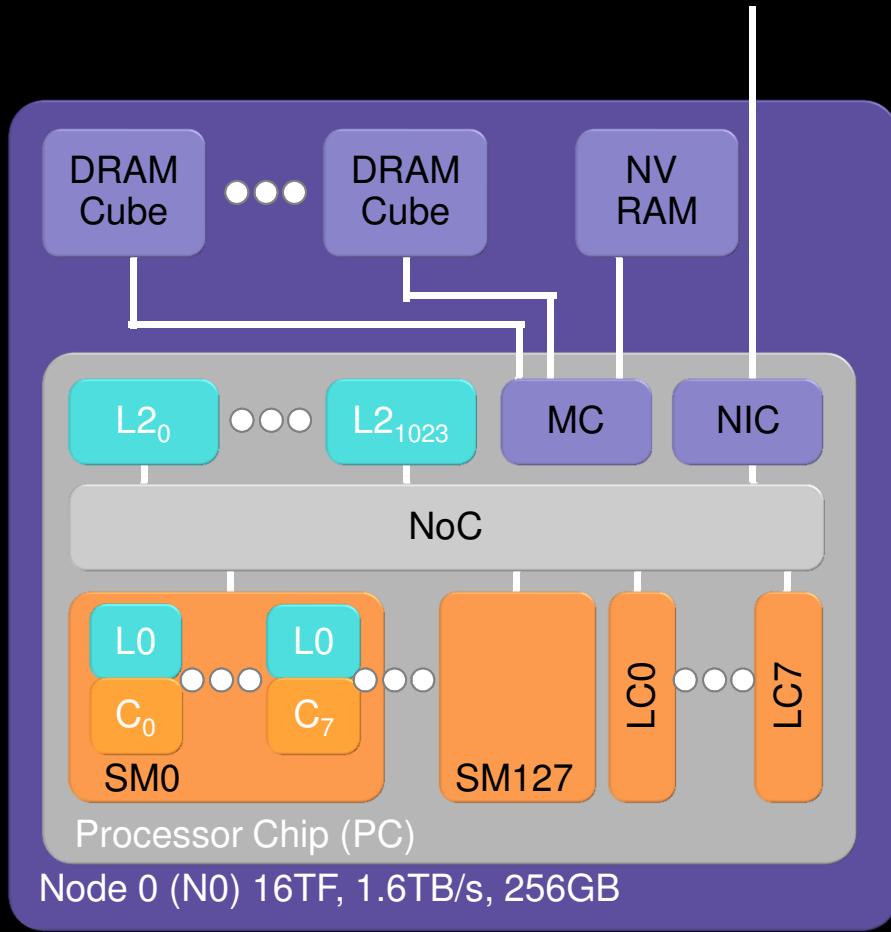
Scalable Parallel Processing

Throughput-optimized
cores

Echelon System Sketch



Echelon Processor Sketch



- Global address space
- Flexible memory hierarchy
- Efficient bulk parallelism
 - thread array creation & mgmt
 - thread synchronization
- **Heterogeneous cores**
 - latency-optimized (LC)
 - throughput-optimized (SM)

Languages must evolve



Hardware trajectory	Mainstream languages
Abundant parallelism	Minimal parallelism (if any)
Heterogeneous cores	Homogeneous core(s)
Memory is complicated	Memory is simple & flat

This is not just an exascale problem



Here are your power-constrained computers.

Phalanx programming model



**How might a mainstream language like C++ evolve to
embrace the trajectory of hardware?**

Design Goals



- **Unified model for heterogeneous parallel machines**
 - single notation for all processors
 - designed for broad range of machines (current & future)
- **Explicit constructs for parallelism & locality**
 - permit experts to write the programs they want to write
 - provide substrate for higher-level abstractions
- **Extending mainstream C/C++ language**
 - implementable as a library whenever possible
 - potential for broad adoption



Thrust: Higher level abstraction

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/sort.h>
#include <cstdlib.h>

int main(void)
{
    // generate 32M random numbers on the host
    thrust::host_vector<int> h_vec(32 << 20);
    thrust::generate(h_vec.begin(), h_vec.end(), rand);

    // transfer data to the device
    thrust::device_vector<int> d_vec = h_vec;

    // sort data on the device
    thrust::sort(d_vec.begin(), d_vec.end());

    // transfer data back to host
    thrust::copy(d_vec.begin(), d_vec.end(), h_vec.begin());

    return 0;
}
```



Key Ingredients

- Processor heterogeneity explicit in type system
- Place-based machine model
- Tasks composed of hierarchical thread arrays
- Pointer-based global address memory model



Example: SAXPY

```
template <typename Platform>
void saxpy (domain<streaming_group>, thread<Platform> > group,
            float a, int n, ptr<float> x, ptr<float> y)
{
    int i = group.subdomain().index();
    if (i < n) y[i] = a * x[i] + y[i];
}

int main (int argc , char** argv)
{
    main_task self = initialize(argc, argv);
    place M = global_machine(self);

    ptr<float> x = allocate<float>(self , M, n);
    ptr<float> y = allocate<float>(self , M, n);
    . . . fill in data . . .

    event e1 = async(self, M, n)(saxpy<x86>, 2.0f, n, x, y);
    wait (self , e1);

    event e2 = async(self, M, n)(saxpy<sm>, 2.0f, n, x, y);
    wait (self, e2);

    . . . deallocate & end program . . .
}
```



Example: SAXPY

```
template <typename Platform>
void saxpy (domain<streaming_group>, thread<Platform> > group,
            float a, int n, ptr<float> x, ptr<float> y)
{
    int i = group.subdomain().index();
    if (i < n) y[i] = a * x[i] + y[i];
}

int main (int argc , char** argv)
{
    main_task self = initialize(argc, argv);
    M = global_malloc(self);
    . . .
    ptr<float> x = allocate<float>(self , M, n);
    ptr<float> y = allocate<float>(self , M, n);
    . . . fill in data . . .

    event e1 = async(self, M, n)(saxpy<x86>, 2.0f, n, x, y);
    wait (self , e1);

    event e2 = async(self, M, n)(saxpy<sm>, 2.0f, n, x, y);
    wait (self, e2);

    . . . deallocate & end program . . .
}
```



Prototype is a template library

```
#include <phalanx/phalanx.h>  
  
using namespace phalanx;
```

Only other requirements:

- compile with nvcc for CUDA support
- link with necessary runtime (e.g., OpenMP)



Library structure

Phalanx C++ program

```
template <typename Platform>
void saxpy (domain<streaming_group>, thread<Platform> > group,
            float a, int n, ptr<float> x, ptr<float> y)
{
    int i = group.subdomain().index();
    if (i < n) y[i] = a * x[i] + y[i];
}

int main (int argc , char** argv)
{
    main_task self = initialize(argc, argv);
    place M = global_machine(self);

    ptr<float> x = allocate<float>(self , M, n);
    ptr<float> y = allocate<float>(self , M, n);
    . . . fill in data . . .

    event e1 = async(self, M, n) (saxpy<x86>, 2.0f, n, x, y);
    wait (self , e1);

    event e2 = async(self, M, n) (saxpy<xsmm>, 2.0f, n, x, y);
    wait (self, e2);

    . . . deallocate & end program . .
}
```

Other libraries
BLAS, Trilinos, Thrust, etc.

Phalanx
generic functions mapping to
multiple backend runtimes

CUDA

OpenMP

GASNet

Fermi
GPU

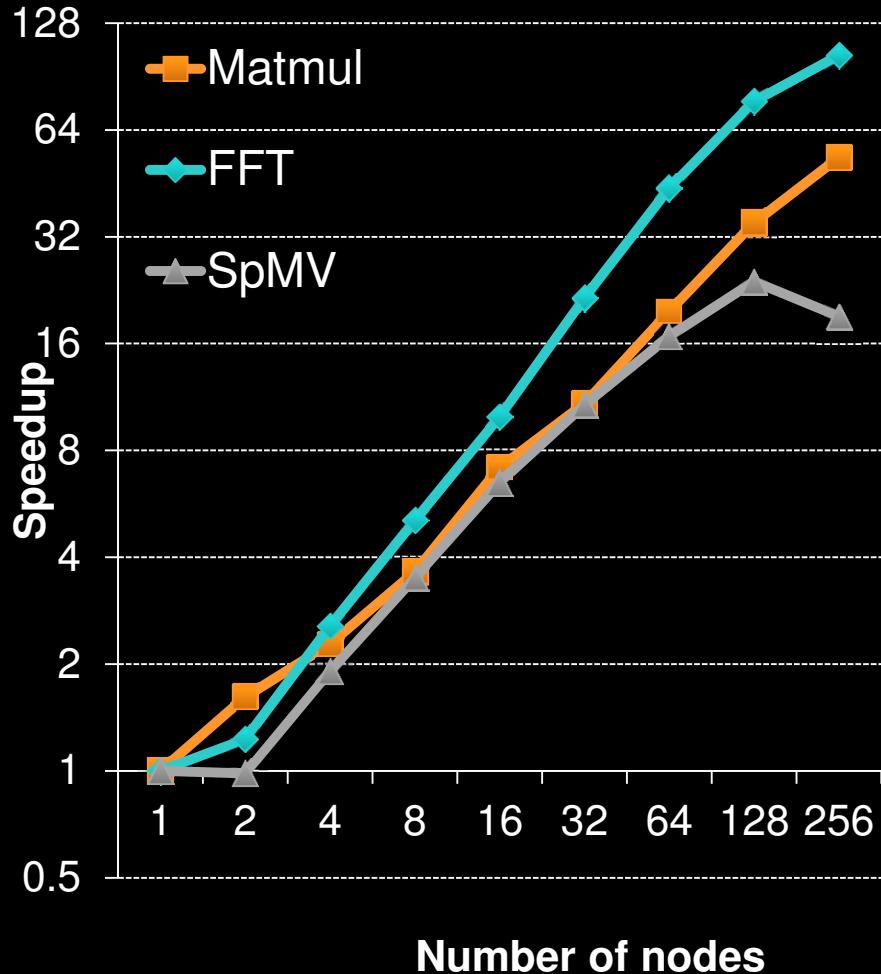
x86
CPU

Multi-node

Library structure



Weak Scaling on Cray XK6



Phalanx C++ program

```
template <typename Platform>
void saxpy (domain<streaming_group>, thread<Platform> > group,
            float a, int n, ptr<float> x, ptr<float> y)
{
    int i = group.subdomain().index();
    if (i < n) y[i] = a * x[i] + y[i];
}

int main (int argc , char** argv)
{
    main_task self = initialize(argc, argv);
    place M = global_machine(self);

    ptr<float> x = allocate<float>(self , M, n);
    ptr<float> y = allocate<float>(self , M, n);
    . . . fill in data . . .

    event e1 = async(self, M, n) (saxpy<x86>, 2.0f, n, x, y);
    wait (self , e1);

    event e2 = async(self, M, n) (saxpy<smm>, 2.0f, n, x, y);
    wait (self , e2);

    . . . deallocate & end program . .
}
```

Phalanx

GASNet

Multi-node



Program knows who & where it is

```
int main(int argc, char **argv)
{
    // Initialize Phalanx runtime
    main_task self = initialize(argc, argv);

    // Get description of machine
    place machine = global_machine(self);

    // Rest of program goes here.
}
```

Every running thread has a task descriptor that:

1. identifies the task, and
2. mediates access to its execution environment

Program knows who & where it is



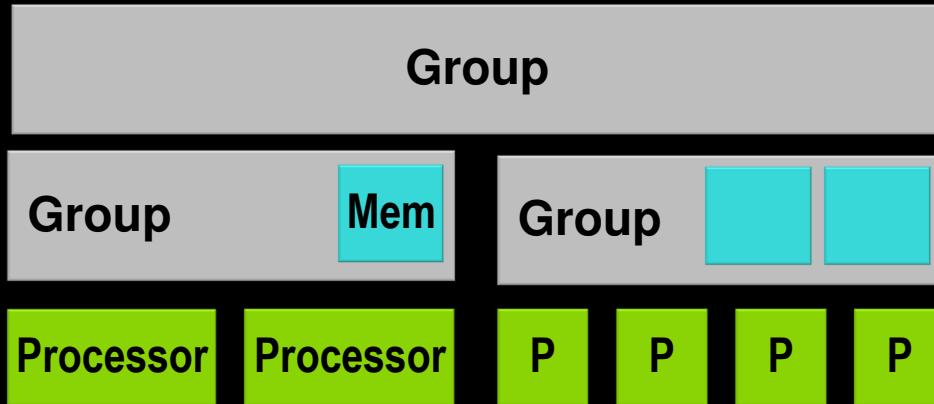
```
int main(int argc, char **argv)
{
    // Initialize Phalanx runtime
    main_task self = initialize(argc, argv);

    // Get description of machine
    place machine = global_machine(self);

    // Rest of program goes here.
}
```

Program has access to “place” objects that name parts of the machine on which they are executing.

Places model hierarchical machines



- **Hierarchical machine**
 - processors at the leaves
 - nodes have 0 or more attached memories
- **Places convey location**
 - tasks launched at places
 - data stored at places

Example interfaces:

```
place M = global_machine(...);  
  
int nodes = M.child_count();  
  
place node0 = M.child(0);  
place mem0 = node0.memories();  
  
place locs = filter_by_isa<arm>(M);  
place tocs = filter_by_isa<sm>(M);
```



Controlling location with places

```
// Get description of machine  
place machine = global_machine(self);
```

```
→ // Allocate space for N integers anywhere  
int *x = allocate<int>(self, machine, N);
```

```
// Allocate N integers on processor P  
int *y = allocate<int>(self,  
    machine.processor(P), N);
```

Specifying the whole machine as a target gives discretion to runtime to place the data.



Controlling location with places

```
// Get description of machine  
place machine = global_machine(self);  
  
// Allocate space for N integers anywhere  
int *x = allocate<int>(self, machine, N);  
  
// Allocate N integers on processor P  
int *y = allocate<int>(self,  
                         machine.processor(P), N);
```

Programs can also be more explicit.



Naming the performing task

```
// Get description of machine
place machine = global_machine(self);

// Allocate space for  $N$  integers anywhere
int *x = allocate<int>(self, machine, N);

// Allocate  $N$  integers on processor  $P$ 
int *y = allocate<int>(self,
                      machine.processor(P), N);
```

Tasks and API functions take a parameter naming the performing task



Type-directed heterogeneity

- Processor heterogeneity should be explicit
- Heterogeneity should be exposed in the type system

```
void task_entry(thread<x86> who) { }
```

```
void task_entry(thread<sm> who) { }
```

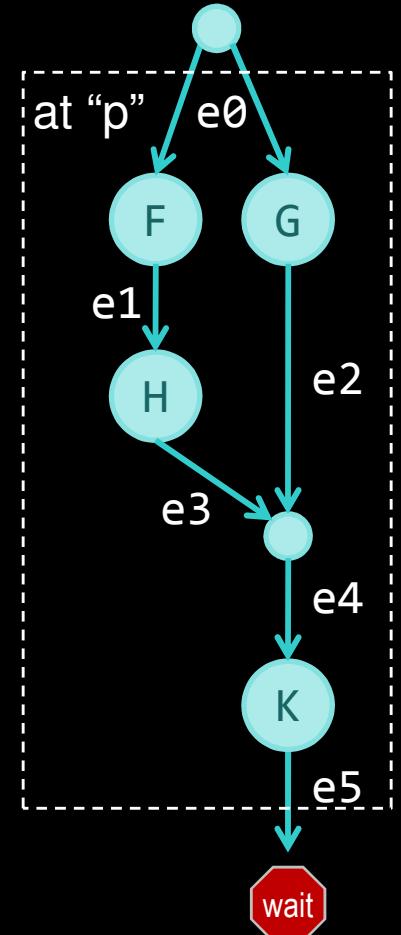
```
template<typename Platform>
void task_entry(thread<Platform> who) { }
```

Asynchronous task graphs



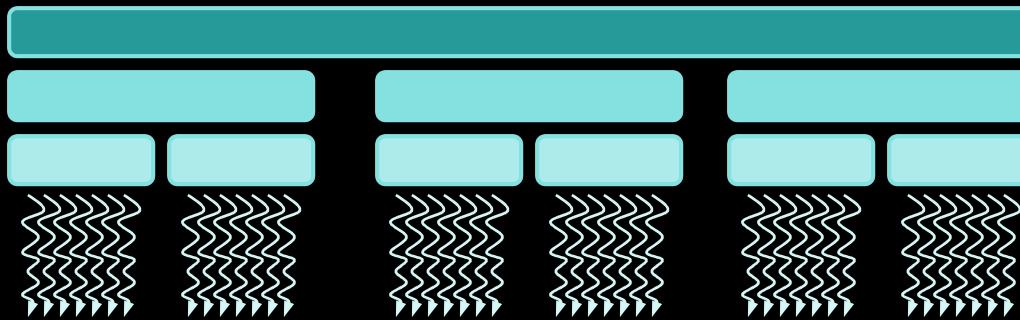
```
template<typename Task>
void do_tasks(Task who, place p, async_event& e0)
{
    async_event e1 = async_after(who, p, e0, dom)(F, ...);
    async_event e2 = async_after(who, p, e0, dom)(G, ...);
    async_event e3 = async_after(who, p, e1, dom)(H, ...);
    async_event e4 = e2 & e3;
    async_event e5 = async_after(who, p, e4, dom)(K, ...);
    wait(here, e5);
}
```

- **Each `async()` statement creates a new task**
- **Events express task ordering dependencies**
- **Scheduler may execute DAG of tasks in any valid order**



Hierarchical threads arrays

- Each task is a hierarchically organized set of threads



- Tasks execute in a SPMD style

```
template<typename Domain>
void entry(Domain task_domain)
{
    int i = task_domain.thread_index();
}
```



Pointer model

- **Pointer types** encode memory space in which they live
 - in addition to the type of data
 - memory spaces indicate semantics of pointers
 - e.g., where different code generation is required
- **Pointer objects** track place to which they point
 - allows program to query location of data

```
ptr<T, MemorySpace> p(address, place);  
place location = p.place();
```

- Place-aware memory allocation / deallocation

```
ptr<T,M> allocate<T,M>(Task who,  
                           place p, int count);  
void deallocate(Task who, ptr<T,M> p);
```

Trends for languages to embrace



- Processor heterogeneity
- Massive processor-level parallelism
- Richer memory hierarchies



Key Phalanx features

- Type-directed heterogeneity
- Hierarchical machine model of “places”
- Asynchronous tasks + hierarchical thread arrays

Further details



Designing a unified programming model for heterogeneous machines

Michael Garland
NVIDIA

Manjunath Kudlur
NVIDIA

Yili Zheng
LBL

To appear in SC12

This research was funded in part by the DARPA UHPC program
under contract HR0011-10-9-0008.



Questions?

mgarland@nvidia.com

<http://research.nvidia.com>