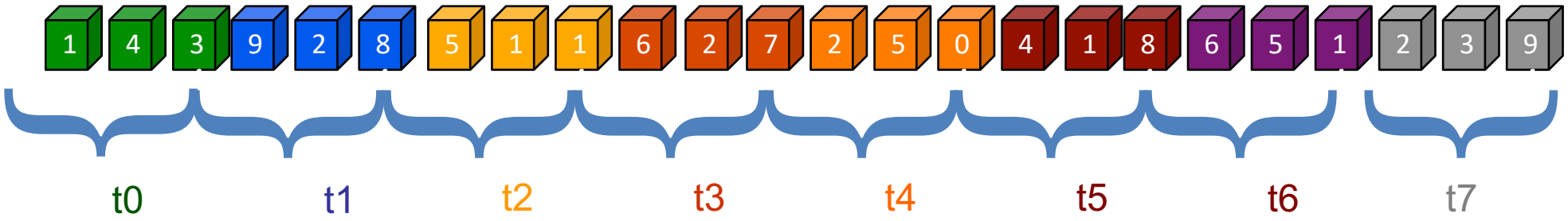
Lecture 5:
Shared--Memory Programming with Pthreads

Beakal Gizachew Assefa

Recap Lecture 4: Parallel Sum

- Computation Partitioning
  - Suppose each task computes a partial sum on n/t consecutive elements (t is the number of tasks)
  - Example: n = 24 and t = 8 tasks



- Workers to Task Mapping
  - Assume we have 8 cores/processors
  - Each worker/thread gets a task
  - Need to calculate the start index for each thread

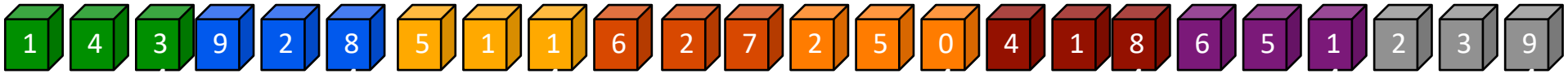
Lecture 4: Data Dependencies

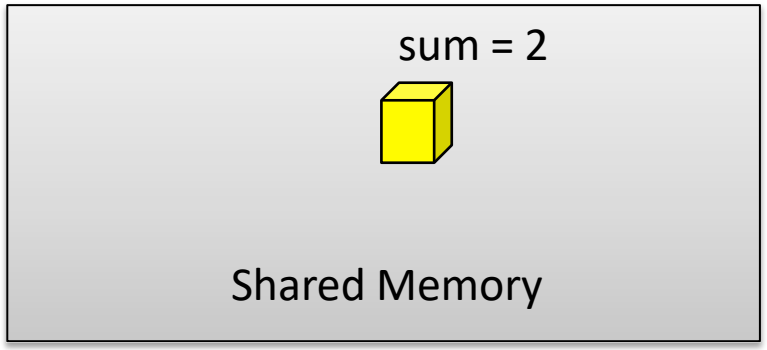
- One of the difficulties of parallel programming comes from the data dependencies between tasks
- Parallel execution has to obey the data dependencies otherwise we will end up with an incorrect program

- A formal definition:
  - *A* ***data dependence*** *is an ordering on a pair of memory operations that must be preserved to maintain correctness.*

Race Condition

- The value of sum is non--deterministic

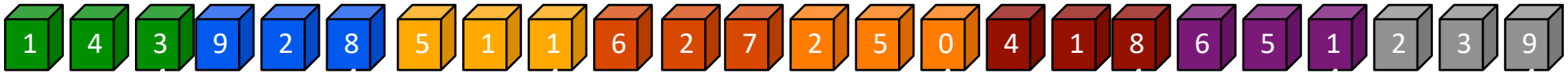


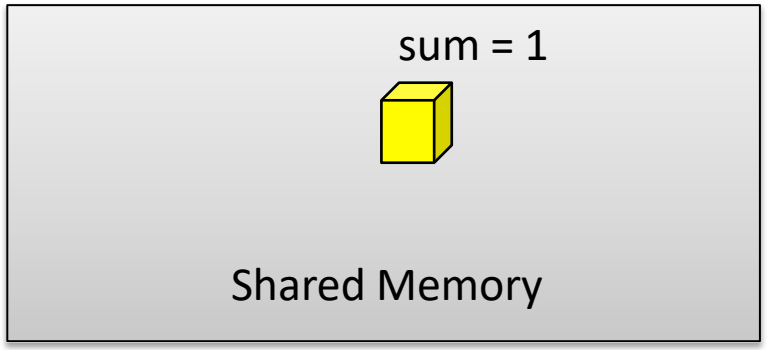


Thread 1	Thread 2		Integer value
			0
read value		←	0
increase value			0
write back		→	1
	read value	←	1
	increase value		1
	write back	→	2

Race Condition

- The value of sum is non--deterministic





Thread 1	Thread 2		Integer value
			0
read value		←	0
	read value	←	0
increase value			0
	increase value		0
write back		→	1
	write back	→	1

Synchronization

- Sum is incorrect if `master' thread begins accumulating final result before other threads are done
- ***Synchronization*** is used to sequence control among threads or to sequence accesses to data in parallel code.
  - Mutual exclusion **(mutex)** so that only one thread at a time is loading/incrementing/storing a memory location **atomically**
  - A ***barrier*** is used to block threads from proceeding beyond a program point until all of the participating threads has reached the barrier.

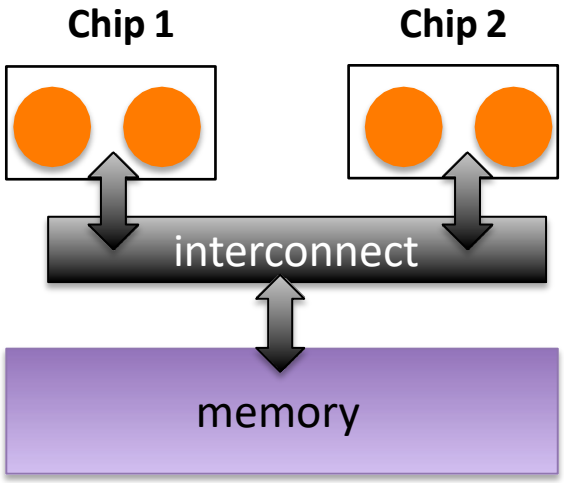
Today

- Shared Memory Programming with Pthreads
- Next Week
  - Shared Memory Programming with OpenMP

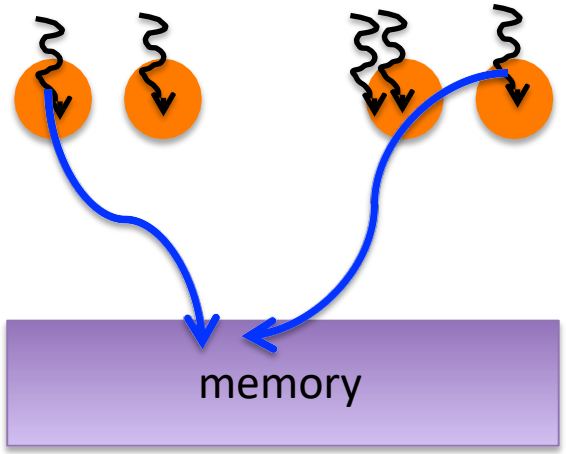
- Readings
  - Pthreads: Book Chapter 4.1-4.3, 4.6
  - Pthreads Tutorial
    - https://computing.llnl.gov/tutorials/pthreads/

Shared--Memory Programming Model

- More correct name: Shared--address space programming
  - Threads communicate through shared memory as opposed to messages
  - Threads coordinate through synchronization (also through shared memory).

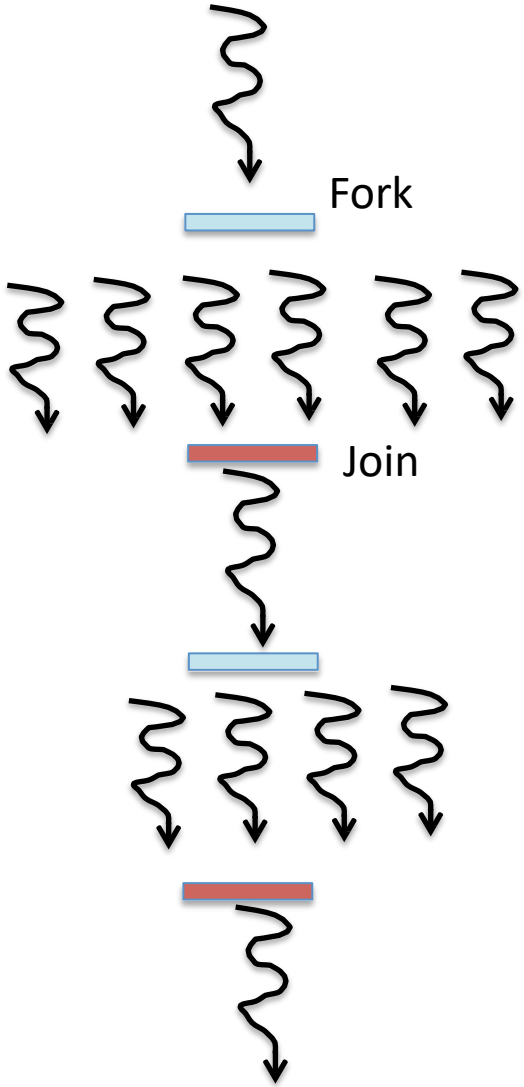


Recall shared memory system
(can be either UMA, NUMA)



Shared--Memory Programming with Threads

- Start with a single root/master thread
- **Fork-join** parallelism to create concurrently executing threads
- A spawned thread executes asynchronously until it completes its task
- Threads may or may not execute on different processors/cores



Programming with Threads

- Several thread libraries out there
  - Pthreads, OpenMP, TBB, Cilk, Qthreads, C++11
- Pthreads is the POSIX (Portable Operating System Interface for Unix) Thread Library
  - Very low level of multi--threaded programming
  - Most widely used for systems--oriented code
- OpenMP is newer standard as compared to Pthread
  - High level support for parallel programming on shared memory

pthreads

- pthreads contain support for
  - Creating parallelism
  - Synchronization
  - No explicit support for communication, because shared memory is implicit; a pointer to shared data is passed to a thread

Thread Creation

- Forking Pthreads

Signature:

```
int pthread_create(pthread_t *,
                   const pthread_attr_t *,
                   void * (*)(void *),
                   void *);
```

Example call:

```
errcode = pthread_create(&thread_id, &thread_attribute,
                         &thread_func, &func_arg);
```

- thread_id is the thread id or handle (used to halt, etc.)
- thread_attribute various attributes
  - standard default values obtained by passing a NULL pointer
- thread_func the function to be run (takes and returns void*)
- func_arg an argument can be passed to thread_fun when it starts
- errorcode will be set to nonzero if the create operation fails

Thread Creation

- The effect of pthread_create
  - Master thread actually causes the operating system to create a new thread
  - Each thread executes a specific function, thread_func
    - The same thread function is executed by all threads that are created, representing the thread's ***computation decomposition***

- For the program to perform different work in different threads, the arguments passed at thread creation distinguish the thread's "id" and any other unique features of the thread.

Simple Example

```
int main() {
  pthread_t threads[16];
  int tn;
  for(tn=0; tn<16; tn++) {
    pthread_create(&threads[tn], NULL, ParFun, NULL);
  }
  for(tn=0; tn<16 ; tn++) {
    pthread_join(threads[tn], NULL);
  }
  return 0;
}
```

- This code creates 16 threads that execute the function “ParFun”.
- Note that thread creation is costly, so it is important that ParFun do a lot of work in parallel to amortize this cost.

Joining Threads

- From Unix specification: “suspends execution of the calling thread until the target thread terminates, unless the target thread has already terminated.”
  - The second parameter allows the exiting thread to pass information back to the calling thread (often NULL).
- Returns nonzero if there is an error

```
int pthread_join(pthread_t *, void **value_ptr)
```

“Hello World”

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int thread_count; //accessible by all threads

void *hello(void* rank); //thread function

int main() {
  pthread_t threads[8]; //thread handles

  int tn;
  for(tn=0; tn<8; tn++) {
    pthread_create(&threads[tn], NULL, hello,(void*)tn);
  }
  for(tn=0; tn<16 ; tn++) {
    pthread_join(threads[tn], NULL);
  }
  return 0;
}
```

declares the various Pthreads functions, constants, types, etc.

Start function to execute

Arguments to function e.g. Thread ID

``Hello World”

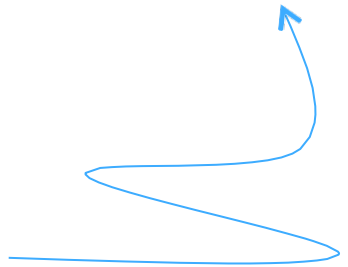
```
void *hello(void* rank){

  int my_rank = (int) rank;

  printf(“Hello from thread %d of %d\n”, my_rank,
                                        thread_count);

  return NULL;
}
```

- Using rank, different execution for each thread is possible

Compiling a Pthread Program

gcc −o pth_hello pth_hello.c −lpthread

link in the Pthreads library



The Pthreads API is only available on POSIX systems
— Linux, MacOS X, Solaris, HPUX, …

Global Variables

- Variables declared outside of ‘main’ are global variables
  - Those variables are shared by all threads
  - Can introduce subtle and confusing bugs
  - Limit the use of global variables to situations in which they are really needed

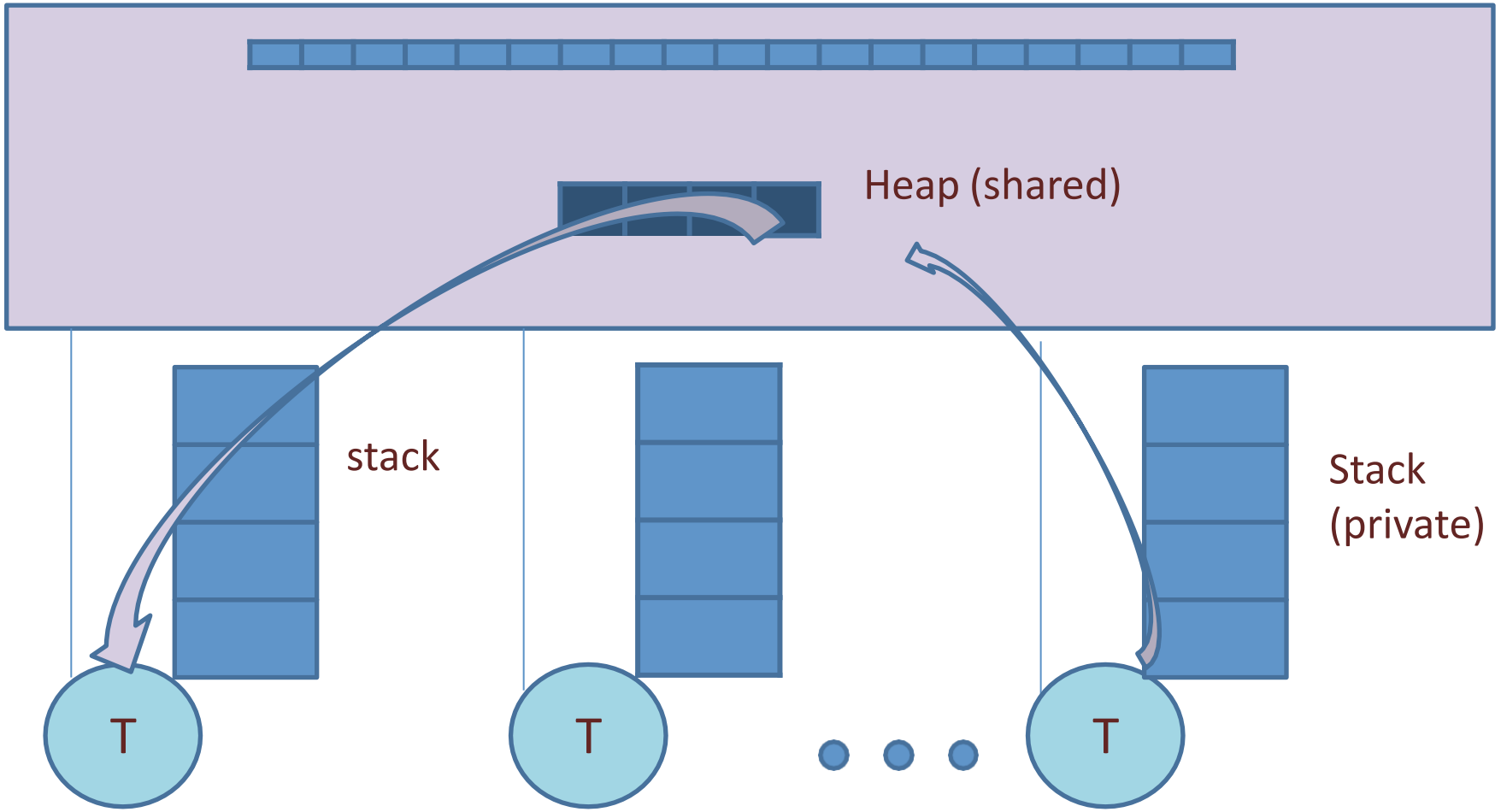
Shared Data

- Object allocated on the **heap** may be shared (if pointer is passed)
- Variables on the **stack are private**: passing pointer to these around to other threads can cause problems

- Shared data often a result of creating a large "thread data" struct
  - Passed into all threads as argument
  - Simple example:

```
char *message = "Hello World!\n";
pthread_create( &thread1,
        NULL,
        (void*)&print_fun,
        (void*) message);
```

Shared Data



- Threads have a private stack but share heap and global data

Synchronization

- Pthread provides several ways to synchronize threads
- Mutexes (mutual exclusion)
  - Used to guarantee that one thread “excludes” all other threads while it executes a section.
- Barriers
  - Synchronizing the threads to make sure that they all are at the same point in a program is called a barrier.
- Semaphores and conditional variables
  - We won’t cover these, they are used implicitly in the underlying implementations but these details are not needed to get started with parallel programming, they are very low level

Mutex (Locks) in Pthreads

```
#include <pthread.h>
pthread_mutex_t amutex = PTHREAD_MUTEX_INITIALIZER;
//create a mutex
pthread_mutex_init(&amutex, NULL);

//use it
pthread_mutex_lock(amutex);
Do_some_quick_work;
pthread_mutex_unlock(amutex);

//destroy a mutex
pthread_mutex_destroy(pthread_mutex_t *mutex);
```

Using Mutex

- Recall parallel sum function from Lecture 4

```
for (i=start; i<start + items_per_task; i++)  {
     my_x = Compute_next_value(…);
     my_sum += my_x;
}
pthread_mutex_lock(&our_lock);
sum+= my_sum;
pthread_mutex_unlock(&our_lock);
```

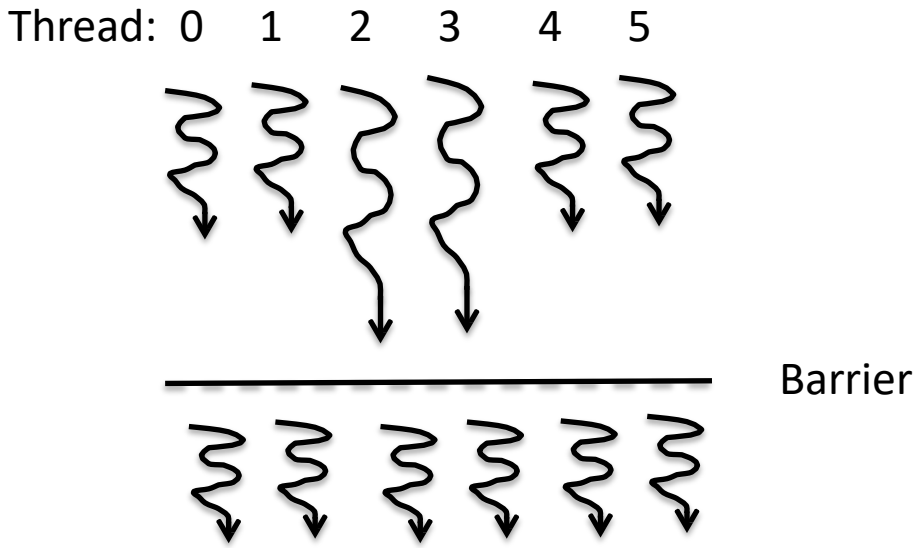
Deadlock

- Multiple mutexes may be held, but can lead to deadlock:
- Thread 1 is waiting for thread 2 to release lock_b, while thread 2 is waiting for thread 1 to release lock_a

thread1	thread2
lock(a)	lock(b)
lock(b)	lock(a)

Barriers

- Synchronizing the threads to make sure that they all are at the same point in a program is called a barrier.
- No thread can cross the barrier until all the threads have reached it.



Even though threads 2 and 3 reached barrier, they will wait for others to arrive. Then all threads cross the barrier point togather.

Barriers in Pthreads

- To (dynamically) initialize a barrier, use code similar to this (which sets the number of threads to 3):

```
pthread_barrier_t b;
pthread_barrier_init(&b,NULL,3);
```

- The second argument specifies an object attribute; using NULL yields the default attributes.
- To wait at a barrier, a process executes:

```
pthread_barrier_wait(&b);
```

- To destroy a barrier

```
pthread_barrier_destroy(&b);
```

Summary of Pthreads

- Pthreads are based on OS features
  - Familiar language for most programmers, particularly for systems people
  - Ability to shared data is convenient
- Pitfalls
  - Very low level
  - Thread management is left to the programmer, introduces bugs
    - Data races and deadlocks
- **OpenMP** is commonly used today as a simpler alternative, but it is more restrictive
  - Next lecture

Thread Safety

- A block of code is thread--safe if it can be simultaneously executed by multiple threads without causing problems.

- Some libraries including C library functions may not be thread--safe!
  - The random number generator random in stdlib.h.
  - The time conversion function localtime in time.h.

- In some cases, the C standard specifies an alternate, thread–safe, version of a function.
  - ‘re--entrant’ (thread--safe) version of the function
  - E.g strtok_r ()

Acknowledgments

- These slides are inspired and partly adapted from
  - Didem Unat (Koc University)
  - Mary Hall (Univ. of Utah)
  - The course book (Pacheco)
  - Pthread Slides: Jim Demmel and Kathy Yelick from UC Berkeley