Shared Memory and OpenMP

Abhinav Bhatele, Department of Computer Science

UNIVERSITY OF MARYLAND

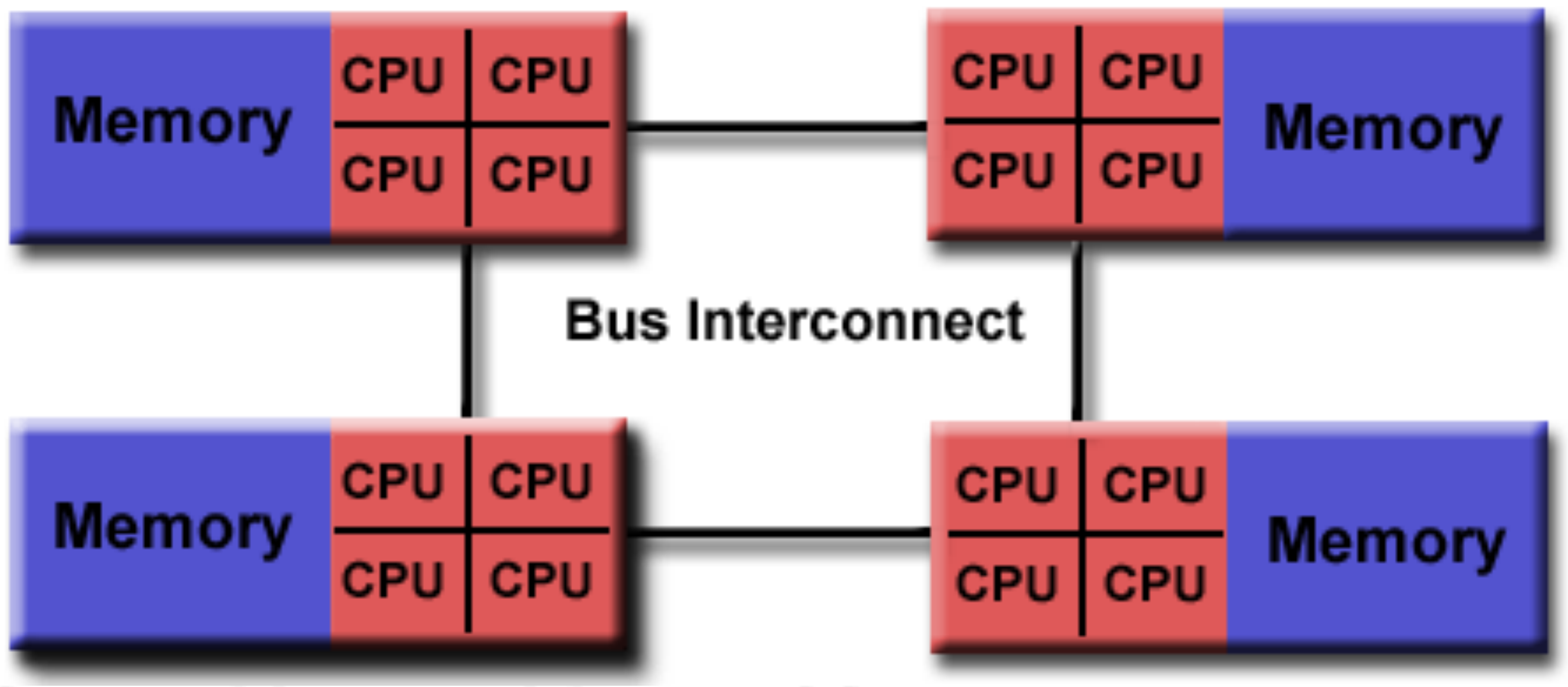
Announcements

- Reminder: Assignment 1 is due on: Feb 28, 11:59 pm ET
- Good-faith attempt of each assignment is required
- Assignment 2 is posted and due on: March 6 at 11:59 pm ET
- Quiz 1 is posted on ELMS, due on March 1 at 11:00 am ET
- Reminders:
  - How to contact course staff: cmsc416-bhatele@cs.umd.edu
  - Do not run/execute code on the login node
  - Best way to report issues: What command did you run, and the full error

Shared memory programming

- All entities (threads) have access to the entire address space
- Threads "communicate" or exchange data by directly accessing shared variables
- Programmer has to manage data conflicts

NUMA Multi-core Node



OpenMP

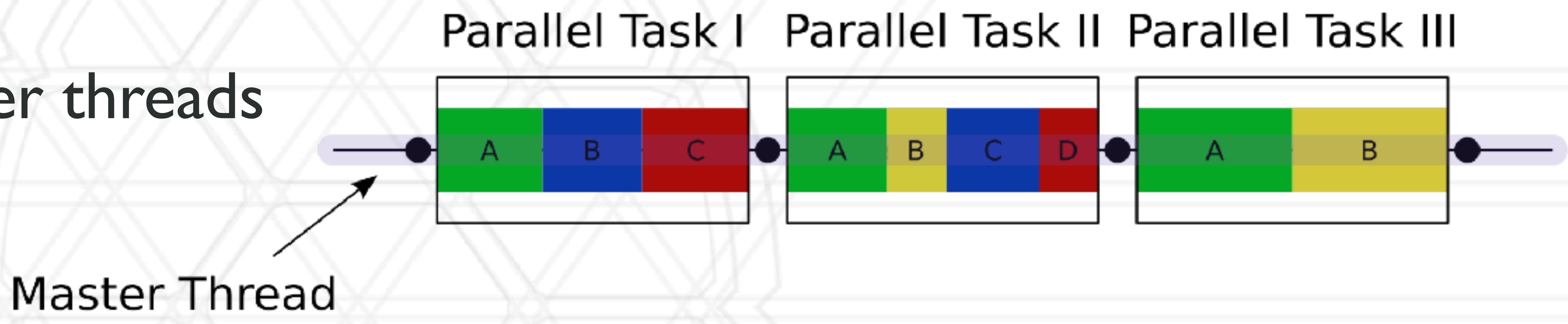
- OpenMP is an example of a shared memory programming model
- Provides within-node parallelization
- Meant for certain kinds of programs/computational kernels
  - Specifically ones that use arrays and loops
- Potentially easier to implement programs in parallel using OpenMP with small code changes (as opposed to distributed memory programming models)

OpenMP

- OpenMP is a language extension (and library) that enables parallelizing C/C++/ Fortran code
- Programmer uses compiler directives and library routines to indicate parallel regions in the code and how to parallelize them
- Compiler converts code to multi-threaded code
- OpenMP uses a fork/join model of parallelism

Fork-join parallelism

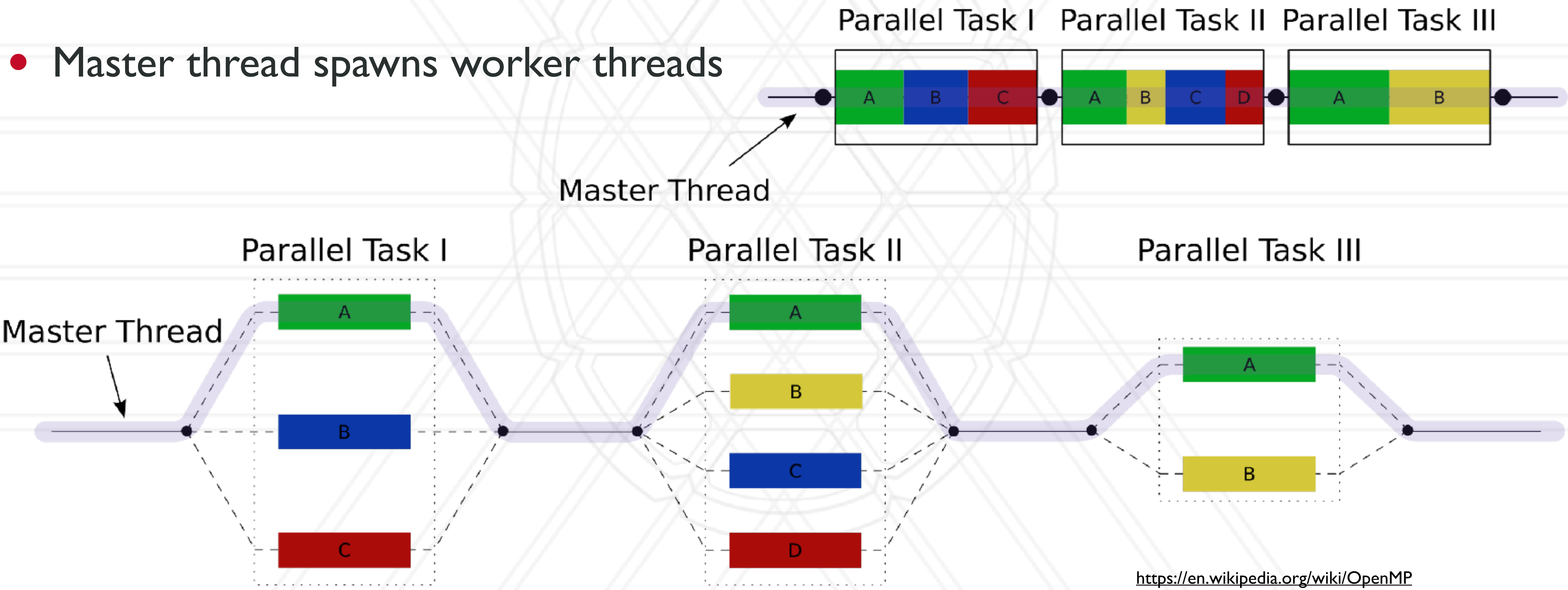
- Single flow of control
- Master thread spawns worker threads



https://en.wikipedia.org/wiki/OpenMP

Fork-join parallelism

- Single flow of control
- Master thread spawns worker threads



https://en.wikipedia.org/wiki/OpenMP

Race conditions when threads interact

- Unintended sharing of variables can lead to race conditions
- Race condition: program outcome depends on the scheduling order of threads
  - More than one thread access a memory location and at least one of them writes to it (without proper synchronization)
- We want program outcome to be deterministic and same as serial program
- How can we prevent data races?
  - Use synchronization
  - Change how data is stored

OpenMP pragmas

- Pragma: a compiler directive in C or C++
- Mechanism to communicate with the compiler
- Compiler may ignore pragmas

```
#pragma omp construct [clause [clause] ... ]
```

Hello World in OpenMP

```
#include <stdio.h>
#include <omp.h>

int main(void)
{
    #pragma omp parallel
    printf("Hello, world.\n");
    return 0;
}
```

- Compiling: `gcc -fopenmp hello.c -o hello`
- Setting number of threads: `export OMP_NUM_THREADS=2`

Parallel for

- Directs the compiler that the immediately following `for` loop should be executed in parallel
- Only applies to the immediately following `for` loop even if you have nested `for` loops

```
#pragma omp parallel for [clause [clause] ... ]
for (i = init; test_expression; increment_expression) {
    ...
    do work
    ...
}
```

Parallel for example

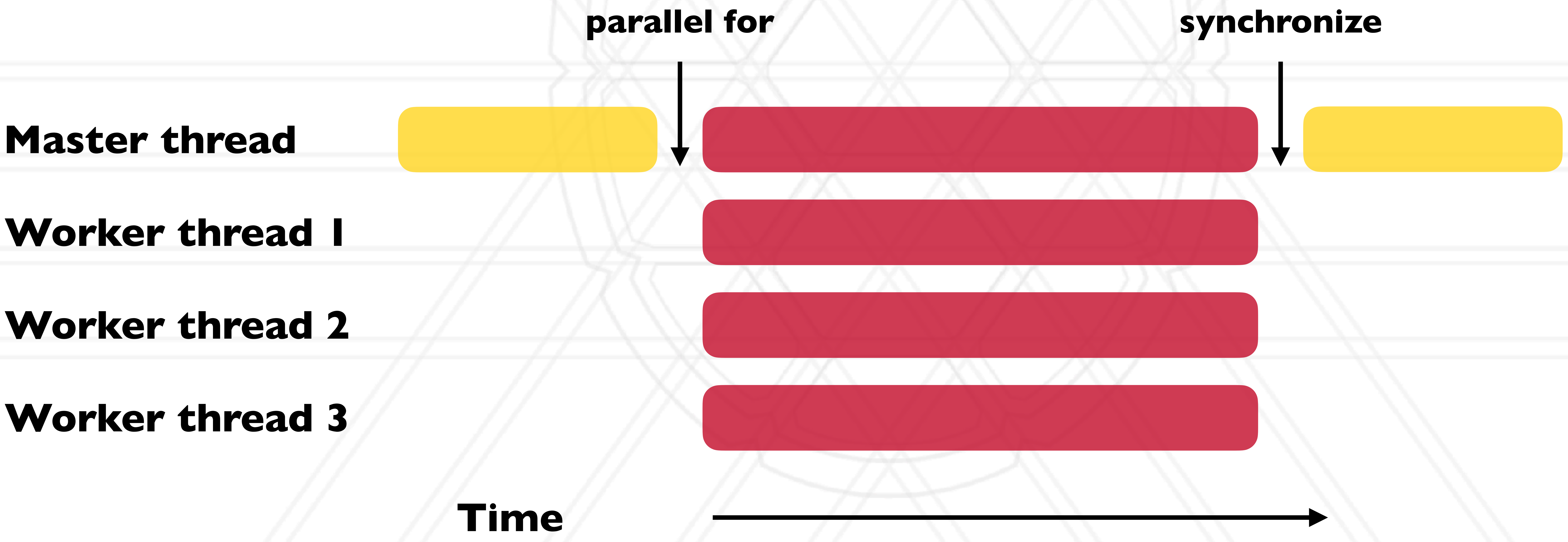
```
int main(int argc, char **argv)
{
    int a[100000];

    #pragma omp parallel for
    for (int i = 0; i < 100000; i++) {
        a[i] = 2 * i;
    }

    return 0;
}
```

Parallel for execution

- Master thread creates worker threads
- The OpenMP runtime distributes iterations of the loop to different threads



Number of threads

- You can set it using this environment variable before executing the program:

```
export OMP_NUM_THREADS=X
```

- From within the program, you can call this library routine:
  - `void omp_set_num_threads(int num_threads)`
    - Set the number of OpenMP threads to be used in parallel regions
- This returns the number of available hardware cores on the node:
  - `int omp_get_num_procs(void);`
    - Can be used to decide the number of threads to create

Data sharing defaults

- Most variables in an OpenMP program are shared by default
- Global variables are shared
- Exception: loop index variables are private by default
- Exception: Stack variables in function calls from parallel regions are also private to each thread (thread-private)

saxpy (single precision a*x+y) example

```
for (int i = 0; i < n; i++) {
    z[i] = a * x[i] + y[i];
}
```

saxpy (single precision a*x+y) example

```
#pragma omp parallel for
for (int i = 0; i < n; i++) {
    z[i] = a * x[i] + y[i];
}
```

Overriding defaults using clauses

- Specify how data is shared between threads executing a parallel region
- `private(list)`
- `shared(list)`
- `default(shared | none)`
- `reduction(operator: list)`
- `firstprivate(list)`
- `lastprivate(list)`

https://www.openmp.org/spec-html/5.0/openmpsu106.html#x139-5540002.19.4

private clause

- Each thread has its own copy of the variables in the list
- Private variables are uninitialized when a thread starts
- The value of a private variable is unavailable to the master thread after the parallel region has been executed

default clause

- Determines the data sharing attributes for variables for which this would be implicitly determined otherwise
- Possible values: `shared` or `none`

Anything wrong with this example?

```
val = 5;

#pragma omp parallel for private(val)
for (int i = 0; i < n; i++) {
    ... = val + 1;
}
```

Anything wrong with this example?

```
val = 5;

#pragma omp parallel for private(val)
for (int i = 0; i < n; i++) {
    ... = val + 1;
}
```

The value of val will not be available to threads inside the loop

Anything wrong with this example?

```
#pragma omp parallel for private(val)
for (int i = 0; i < n; i++) {
    val = i + 1;
}

printf("%d\n", val);
```

Anything wrong with this example?

```
#pragma omp parallel for private(val)
for (int i = 0; i < n; i++) {
    val = i + 1;
}

printf("%d\n", val);
```

The value of val will not be available to the master thread outside the loop

firstprivate clause

- Initializes each thread's private copy to the value of the master thread's copy upon entry to the parallel section

```
val = 5;

#pragma omp parallel for firstprivate(val)
for (int i = 0; i < n; i++) {
    ... = val + 1;
}
```

lastprivate clause

- Writes the value belonging to the thread that executed the last iteration of the loop to the master's copy
- Last iteration determined by sequential order

lastprivate clause

- Writes the value belonging to the thread that executed the last iteration of the loop to the master's copy
- Last iteration determined by sequential order

```
#pragma omp parallel for lastprivate(val)
for (int i = 0; i < n; i++) {
    val = i + 1;
}

printf(“%d\n”, val);
```

reduction(operator: list) clause

- Reduce values across private copies of a variable
- Operators: +, -, *, &, |, ^, &&, ||, max, min

```
#pragma omp parallel for
for (int i = 0; i < n; i++) {
    val += i;
}

printf(“%d\n”, val);
```

https://www.openmp.org/spec-html/5.0/openmpsu107.html#x140-5800002.19.5

reduction(operator: list) clause

- Reduce values across private copies of a variable
- Operators: +, -, *, &, |, ^, &&, ||, max, min

```
#pragma omp parallel for reduction(+: val)
for (int i = 0; i < n; i++) {
    val += i;
}

printf("%d\n", val);
```

Identifier	Initializer	Combiner
+	omp_priv = 0	omp_out += omp_in
-	omp_priv = 0	omp_out += omp_in
*	omp_priv = 1	omp_out *= omp_in
&	omp_priv = ~ 0	omp_out &= omp_in
|	omp_priv = 0	omp_out |= omp_in
^	omp_priv = 0	omp_out ^= omp_in
&&	omp_priv = 1	omp_out = omp_in && omp_out
||	omp_priv = 0	omp_out = omp_in || omp_out
max	omp_priv = *Least representable number in the reduction list item type*	omp_out = omp_in > omp_out ? omp_in : omp_out
min	omp_priv = *Largest representable number in the reduction list item type*	omp_out = omp_in < omp_out ? omp_in : omp_out

https://www.openmp.org/spec-html/5.0/openmpsu107.html#x140-5800002.19.5

Loop scheduling

- Assignment of loop iterations to different worker threads
- Default schedule tries to balance iterations among threads
- User-specified schedules are also available

User-specified loop scheduling

- Schedule clause

```
schedule (type[, chunk])
```

- type: static, dynamic, guided, runtime
- static: iterations divided as evenly as possible (#iterations/#threads)
  - chunk < #iterations/#threads can be used to interleave threads
- dynamic: assign a chunk size block to each thread
  - When a thread is finished, it retrieves the next block from an internal work queue
  - Default chunk size = 1

Other schedules

- guided: similar to dynamic but start with a large chunk size and gradually decrease it for handling load imbalance between iterations
- auto: scheduling delegated to the compiler
- To set schedule at runtime: use the OMP_SCHEDULE environment variable

https://software.intel.com/content/www/us/en/develop/articles/openmp-loop-scheduling.html

Calculate the value of $\pi = \int_0^1 \frac{4}{1+x^2}$

```
int main(int argc, char *argv[])
{
    ...

    n = 10000;

    h   = 1.0 / (double) n;
    sum = 0.0;

    for (i = 1; i <= n; i += 1) {
        x = h * ((double)i - 0.5);
        sum += (4.0 / (1.0 + x * x));
    }
    pi = h * sum;

    ...
}
```

Calculate the value of $\pi = \int_0^1 \frac{4}{1+x^2}$

```
int main(int argc, char *argv[])
{
    ...

    n = 10000;
    h   = 1.0 / (double) n;
    sum = 0.0;

    #pragma omp parallel for firstprivate(h) private(x) reduction(+: sum)
    for (i = 1; i <= n; i += 1) {
        x = h * ((double)i - 0.5);
        sum += (4.0 / (1.0 + x * x));
    }
    pi = h * sum;

    ...
}
```

Parallel region

- All threads execute the structured block

```
#pragma omp parallel [clause [clause] ... ]
    structured block
```

- Number of threads can be specified just like the parallel for directive

Synchronization

- Concurrent access to shared data may result in inconsistencies
- Use mutual exclusion to avoid that
- critical directive
- atomic directive
- Library lock routines

https://software.intel.com/content/www/us/en/develop/documentation/advisor-user-guide/top/appendix/adding-parallelism-to-your-program/replacing-annotations-with-openmp-code/adding-openmp-code-to-synchronize-the-shared-resources.html

critical directive

- Specifies that the code is only to be executed by one thread at a time

```
#pragma omp critical [(name)]
    structured block
```

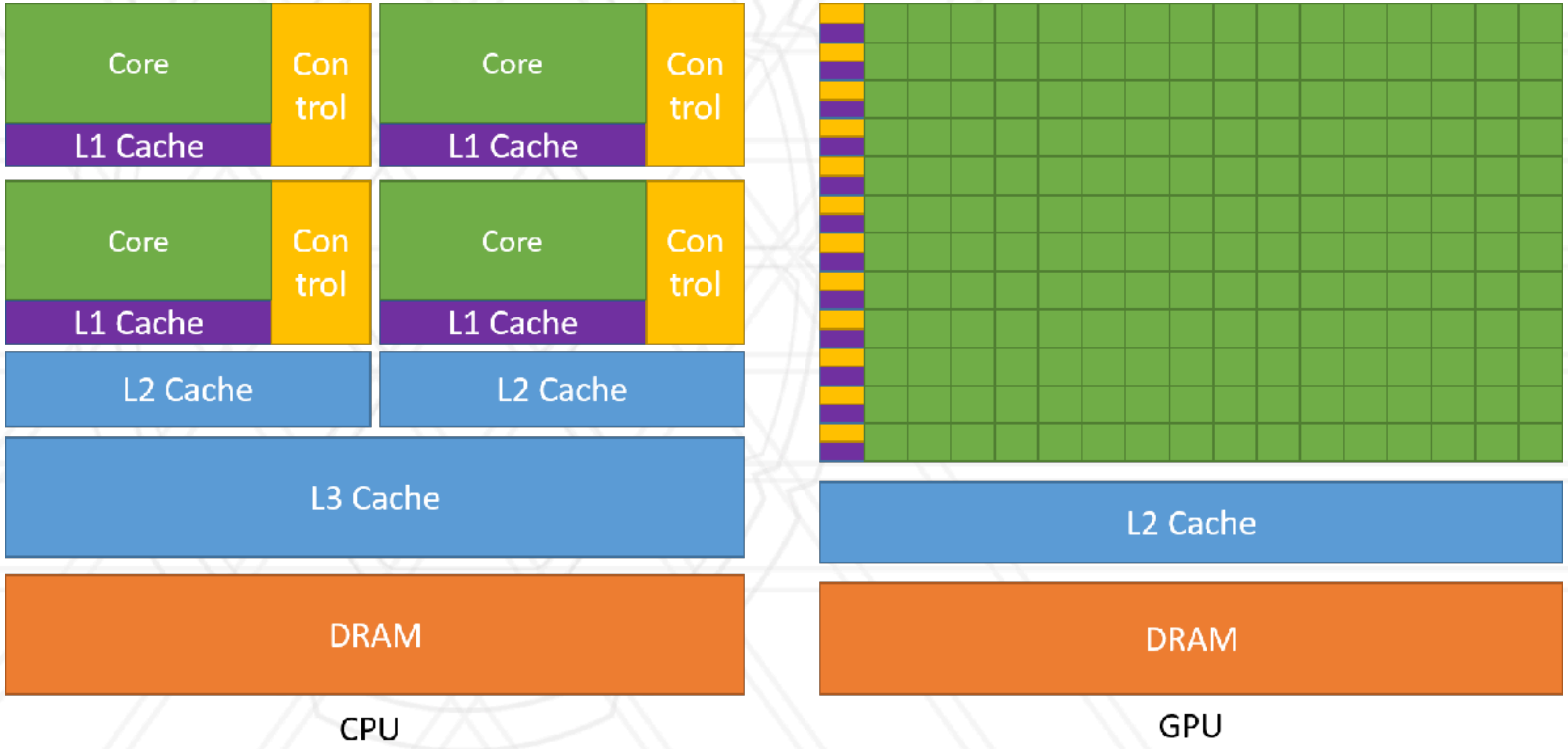
atomic directive

- Specifies that a memory location should be updated atomically

```
#pragma omp atomic
    expression
```

GPGPUs

- GPGPU: General Purpose Graphical Processing Unit
- Many slower cores



https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

OpenMP on GPUs

- *target*: run on accelerator / device

```
for (int i = 0; i < n; i++) {
    z[i] = a * x[i] + y[i];
}
```

- *teams distribute*: creates a team of worker threads and distributes work amongst them

OpenMP on GPUs

- *target*: run on accelerator / device

```
#pragma omp target teams distribute parallel for
for (int i = 0; i < n; i++) {
    z[i] = a * x[i] + y[i];
}
```

- *teams distribute*: creates a team of worker threads and distributes work amongst them

UNIVERSITY OF MARYLAND

Abhinav Bhatele

5218 Brendan Iribe Center (IRB) / College Park, MD 20742

phone: 301.405.4507 / e-mail: bhatele@cs.umd.edu