SWARM Tutorial

Chen Chen

4/12/2012
Outline

• Introduction to SWARM
• Programming in SWARM
• Atomic Operations in SWARM
• Parallel For Loop in SWARM
Outline

• **Introduction to SWARM**
• Programming in SWARM
• Atomic Operations in SWARM
• Parallel For Loop in SWARM
What is SWARM

• SWARM (SWift Adaptive Runtime Machine)
• A **dynamic adaptive runtime system** that minimizes user exposure to physical parallelism and system complexity
• SWARM is designed to enable programming on many-core architectures by utilizing a **dynamic, message-driven model** of execution instead of the static scheduling and sequential computing method of conventional programming models

Difference Between OpenMP and SWARM

<table>
<thead>
<tr>
<th>OpenMP</th>
<th>SWARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse-grain execution model</td>
<td>Fine-grain execution model</td>
</tr>
</tbody>
</table>

### Parallel Tasks
- Wake up all threads
- Barrier

### Sequential Task
- Master thread
- Slave threads
How Codelets Work in SWARM

- Each codelet is attached a dependency counter with some initial positive value
- A codelet is in *dormant* state if its counter is greater than 0
- A codelet is in *enabled* state if its counter is 0
- An *enabled* codelet will be scheduled to a free thread and executed (*firing state*)
- A codelet can call `satisfy()` to decrease the value of any dependency counter
- SWARM runtime handles codelet schedule and dependency maintenance
Suppose we have 3 codelets. Codelet3 cannot start unless both codelet1 and codelet2 are done. And suppose we use one thread to execute the 3 codelets.
How Codelet Works in SWARM – an Example

Suppose we have 3 codelets. Codelet3 cannot start unless both codelet1 and codelet2 are done. And suppose we use one thread to execute the 3 codelets.
How Codelet Works in SWARM – an Example

Suppose we have 3 codelets. Codelet3 cannot start unless both codelet1 and codelet2 are done. And suppose we use one thread to execute the 3 codelets.
How Codelet Works in SWARM – an Example

Suppose we have 3 codelets. Codelet3 cannot start unless both codelet1 and codelet2 are done. And suppose we use one thread to execute the 3 codelets.
How Codelet Works in SWARM – an Example

Suppose we have 3 codelets. Codelet3 cannot start unless both codelet1 and codelet2 are done. And suppose we use one thread to execute the 3 codelets.
Suppose we have 3 codelets. Codelet3 cannot start unless both codelet1 and codelet2 are done. And suppose we use one thread to execute the 3 codelets.
How Codelet Works in SWARM – an Example

Suppose we have 3 codelets. Codelet3 cannot start unless both codelet1 and codelet2 are done. And suppose we use one thread to execute the 3 codelets.
Outline

• Introduction to SWARM
• Programming in SWARM
• Atomic Operations in SWARM
• Parallel For Loop in SWARM
Programming in SWARM

Partition the problem into codelets

Problem → codelets → Classify

- **Entry codelet**: No parent
- **Single dependent codelet**: One parent
- **Multiple dependent codelet**: Multiple parents
- **Sink codelet**: Executed at last
Programming in SWARM cont.

Setup dependency in the program

Call `swarm_enterRuntime()` to start **entry codelet**

**Entry codelet** has no parent. We execute it at beginning of SWARM runtime.
Programming in SWARM cont.

Setup dependency in the program

Call `swarm_enterRuntime()` to start **entry codelet**

Call `swarm_scheduleGeneral()` to create a **single dependent codelet** at the end of its parent

**Single dependent codelet** has only one parent. We create it at the end of its parent without setting dependency.
Programming in SWARM cont.

Setup dependency in the program

Call `swarm_enterRuntime()` to start **entry codelet**

Call `swarm_scheduleGeneral()` to create a **single dependent codelet** at the end of its parent

Call `swarm_dependency_init()` to create a **multiple dependent codelet** before any of its parent is created (e.g., create it at some ancient of all its parents) and setup dependencies

A **multiple dependent codelet** has multiple parents. We have to create it and set its dependency counter. We perform the creation and setting before the start of any of its parents to avoid conflicts in the dependency counter.
Programming in SWARM cont.

Setup dependency in the program

- Call `swarm_enterRuntime()` to start **entry codelet**

- Call `swarm_scheduleGeneral()` to create a **single dependent codelet** at the end of its parent

- Call `swarm_dependency_init()` to create a **multiple dependent codelet** before any of its parent is created (e.g., create it at some ancient of all its parents) and setup dependencies

- **Sink codelet** is created in the same way as either **single dependent codelet** or **multiple dependent codelet**, according to the number of its parents

Call `swarm_shutdownRuntime()` at the end of **sink codelet** to terminate SWARM runtime.
Example: Hello World

Codelet graph of hello world

8 in all

startup

hello

world

hello

world

done
#include <stdio.h>
#include <swarm/Runtime.h>
#include <swarm/Scheduler.h>

#define COUNT 8

static void startup(void *);
static void hello(void *);
static void world(void *);
static void done(void *);

int main(void)
{
    return !swarm_enterRuntime(NULL,
                                startup, NULL);
}

static swarm_dependency_t dep;
static void startup(void *unused)
{
    unsigned i;
    (void)unused;

    swarm_dependency_init(&dep, COUNT, done, NULL);
    for(i=0; i<COUNT; i++)
    {
        swarm_scheduleGeneral(hello, (void *)(size_t)i);
    }

static void hello(void *i)
{
    const unsigned i = (size_t)_i;

    printf("%u: Hello,
           
")
    swarm_scheduleGeneral(world, _i);
}

static void world(void *i)
{
    const unsigned i = (size_t)_i;

    printf("%u: world!
           
")
    swarm_satisfy(&dep, 1);
}

static void done(void *unused)
{
    (void)unused;
    puts("All done!");
    swarm_shutdownRuntime(NULL);
}
Example: Hello World

```c
#include <stdio.h>
#include <swarm/Runtime.h>
#include <swarm/Scheduler.h>

#define COUNT 8

static void startup(void *);
static void hello(void *);
static void world(void *);
static void done(void *);

int main(void)
{
    return !swarm_enterRuntime(NULL,
        startup, NULL);
}

static swarm_dependency_t dep;
static void startup(void *unused)
{
    unsigned i;
    (void)unused;

    swarm_dependency_init(&dep, COUNT, done, NULL);
    for(i=0; i<COUNT; i++)
        swarm_scheduleGeneral(hello, (void *)(size_t)i);
}

static void hello(void *i)
{
    const unsigned i = (size_t)_i;
    printf("%u: Hello,\n", i);
    swarm_scheduleGeneral(world, _i);
}

static void world(void *i)
{
    const unsigned i = (size_t)_i;
    printf("%u: world!\n", i);
    swarm_satisfy(&dep, 1);
}

static void done(void *unused)
{
    (void)unused;
    puts("All done!");
    swarm_shutdownRuntime(NULL);
}
```
Example: Hello World

```c
#include <stdio.h>
#include <swarm/Runtime.h>
#include <swarm/Scheduler.h>

#define COUNT 8

static void startup(void *);
static void hello(void *);
static void world(void *);
static void done(void *);

int main(void)
{
    return !swarm_enterRuntime(NULL, startup, NULL);
}

static swarm_dependency_t dep;
static void startup(void *unused)
{
    unsigned i;
    (void)unused;
    swarm_dependency_init(&dep, COUNT, done, NULL);
    for(i=0; i<COUNT; i++)
        swarm_scheduleGeneral(hello, (void *)(size_t)i);
}

static void hello(void *i)
{
    const unsigned i = (size_t)_i;
    printf("%u: Hello,\n", i);
    swarm_scheduleGeneral(world, _i);
}

static void world(void *i)
{
    const unsigned i = (size_t)_i;
    printf("%u: world!\n", i);
    swarm_satisfy(&dep, 1);
}

static void done(void *unused)
{
    (void)unused;
    puts("All done!");
    swarm_shutdownRuntime(NULL);
}
```
Example: Hello World

```c
#include <stdio.h>
#include <swarm/Runtime.h>
#include <swarm/Scheduler.h>

#define COUNT 8

static void startup(void *);
static void hello(void *);
static void world(void *);
static void done(void *);

int main(void)
{
    return !swarm_enterRuntime(NULL,
                                startup, NULL);
}

static swarm_dependency_t dep;
static void startup(void *unused)
{
    unsigned i;
    (void)unused;

    swarm_dependency_init(&dep, COUNT, done, NULL);
    for(i=0; i<COUNT; i++)
        swarm_scheduleGeneral(hello, (void *)(size_t)i);
}

static void hello(void *i)
{
    const unsigned i = (size_t)_i;
    printf("%u: Hello,\n", i);
    swarm_scheduleGeneral(world, _i);
}

static void world(void *i)
{
    const unsigned i = (size_t)_i;
    printf("%u: world!\n", i);
    swarm_satisfy(&dep, 1);
}

static void done(void *unused)
{
    (void)unused;
    puts("All done!");
    swarm_shutdownRuntime(NULL);
}
```
#include <stdio.h>
#include <swarm/Runtime.h>
#include <swarm/Scheduler.h>

#define COUNT 8

static void startup(void *);
static void hello(void *);
static void world(void *);
static void done(void *

int main(void)
{
    return !swarm_enterRuntime(NULL,
        startup, NULL);
}

static swarm_dependency_t dep;

static void startup(void *)
{
    unsigned i;
    (void)unused;

    swarm_dependency_init(&dep, COUNT, done, NULL);
    for(i=0; i<COUNT; i++)
        swarm_scheduleGeneral(hello, (void *)(size_t)i);
}

static void hello(void *i)
{
    const unsigned i = (size_t)i;
    printf("%u: Hello,\n", i);
    swarm_scheduleGeneral(world, _i);
}

static void world(void *i)
{
    const unsigned i = (size_t)i;
    printf("%u: world!\n", i);
    swarm_satisfy(&dep, 1);
}

static void done(void *unused)
{
    (void)unused;
    puts("All done!");
    swarm_shutdownRuntime(NULL);
}
SWARM APIs – Enter SWARM Runtime

- `swarm_enterRuntime(params, codelet, context)`
  - `params`: pointer to `swarm_Runtime_params_t`
    - Setting up SWARM runtime (e.g., max number of threads)
  - `codelet`: function name (codelet)
    - Function in the format “void fname(void * context)”
    - Entry codelet
  - `context`: pointer to a data structure
    - The parameters passed to the codelet
```
#include <stdio.h>
#include <swarm/Runtime.h>
#include <swarm/Scheduler.h>

#define COUNT 8

static void startup(void *);
static void hello(void *
static void world(void *
static void done(void *

int main(void)
{
    return !swarm_enterRuntime(NULL,
startup, NULL);
}

static swarm_dependency_t dep;
static void startup(void *unused)
{
    unsigned i;
    (void)unused;

    swarm_dependency_init(&dep, COUNT, done, NULL);
    for(i=0; i<COUNT; i++)
        swarm_scheduleGeneral(hello, (void *)(size_t)i);
}

static void hello(void *__i)
{
    const unsigned i = (size_t)__i;
    printf("%u: Hello,\n", i);
    swarm_scheduleGeneral(world, __i);
}

static void world(void *__i)
{
    const unsigned i = (size_t).__i;
    printf("%u: world!\n", i);
    swarm_satisfy(&dep, 1);
}

static void done(void *unused)
{
    (void)unused;
    puts("All done!");
    swarm_shutdownRuntime(NULL);
}
```
SWARM APIs – Codelet Creation (1)

• Create a free codelet, i.e., the codelet does not depend on other codelets
  – `swarm_scheduleGeneral(codelet, context)`
    • `codelet`: function name (codelet)
      – Function in the format “void fname(void * context)"
    • `context`: pointer to a data structure
      – The parameters passed to the codelet
Review Hello World

```c
#include <stdio.h>
#include <swarm/Runtime.h>
#include <swarm/Scheduler.h>

#define COUNT 8

static void startup(void *);
static void hello(void *);
static void world(void *);
static void done(void *);

int main(void)
{
    return !swarm_enterRuntime(NULL,
        startup, NULL);
}

static swarm_dependency_t dep;
static void startup(void *unused)
{
    unsigned i;
    (void)unused;

    swarm_dependency_init(&dep, COUNT, done, NULL);
    for(i=0; i<COUNT; i++)
        swarm_scheduleGeneral(hello, (void *)(size_t)i);
}

static void hello(void *i)
{
    const unsigned i = (size_t)i;
    printf("%u: Hello,\n", i);
    swarm_scheduleGeneral(world, i);
}

static void world(void *i)
{
    const unsigned i = (size_t)i;
    printf("%u: world!\n", i);
    swarm_satisfy(&dep, 1);
}

static void done(void *unused)
{
    (void)unused;
    puts("All done!");
    swarm_shutdownRuntime(NULL);
}
```
SWARM APIs – Codelet Creation (2)

• Create a dependent codelet, i.e., the codelet depends on other codelets
  – swarm_dependency_init(dep, count, codelet, context)
    • dep: pointer to swarm_dependency_t
      – A variable that stores number of satisfied dependencies
    • count: An integer that specifies required dependencies
    • codelet: function name (codelet)
      – Function in the format “void fname(void * context)”
    • context: pointer to a data structure
      – The parameters passed to the codelet
Review Hello World

```c
#include <stdio.h>
#include <swarm/Runtime.h>
#include <swarm/Scheduler.h>

#define COUNT 8

static void startup(void *);
static void hello(void *);
static void world(void *);
static void done(void *);

int main(void)
{
    return !swarm_enterRuntime(NULL,
        startup, NULL);
}

static void hello(void * _i)
{
    const unsigned i = (size_t)_i;
    printf("%u: Hello,
", i);
    swarm_scheduleGeneral(world, _i);
}

static void world(void * _i)
{
    const unsigned i = (size_t)_i;
    printf("%u: world!
", i);
    swarm_satisfy(&dep, 1);
}

static void done(void * unused)
{
    (void)unused;
    puts("All done!");
    swarm_shutdownRuntime(NULL);
}

static void startup(void * unused)
{
    unsigned i;
    (void)unused;

    swarm_dependency_init(&dep, COUNT, done, NULL);
    for(i=0; i<COUNT; i++)
    { 
        swarm_scheduleGeneral(hello, (void *)(size_t)i);
    }
}
```

8 in all
SWARM APIs – Satisfy a Dependency

• `swarm_satisfy(dep, num)`
  – `dep`: pointer to `swarm_dependency_t`
    • A variable that stores number of satisfied dependencies
  – `num`: An integer that specifies the number of times to satisfy `dep`
Review Hello World

```c
#include <stdio.h>
#include <swarm/Runtime.h>
#include <swarm/Scheduler.h>

#define COUNT 8

static void startup(void *);
static void hello(void *);
static void world(void *);
static void done(void *);

int main(void)
{
    return !swarm_enterRuntime(NULL,
            startup, NULL);
}

static swarm_dependency_t dep;
static void startup(void *
 unused)
{
    unsigned i;
    (void)unused;

    swarm_dependency_init(&dep, COUNT, done, NULL);
    for(i=0; i<COUNT; i++)
        swarm_scheduleGeneral(hello, (void *)(size_t)i);
}

static void hello(void *i)
{
    const unsigned i = (size_t)_i;
    printf("%u: Hello,\n", i);
    swarm_scheduleGeneral(world, _i);
}

static void world(void *i)
{
    const unsigned i = (size_t)_i;
    printf("%u: world!\n", i);
    swarm_satisfy(&dep, 1);
}

static void done(void *
 unused)
{
    (void)unused;
    puts("All done!");
    swarm_shutdownRuntime(NULL);
}
```

8 in all

...
SWARM APIs – Terminate SWARM Runtime

• `swarm_shutdownRuntime(NULL)`
  – Shut down the runtime in which the caller is executing
Review Hello World

```c
#include <stdio.h>
#include <swarm/Runtime.h>
#include <swarm/Scheduler.h>

#define COUNT 8

static void startup(void *);
static void hello(void *);
static void world(void *);
static void done(void *);

int main(void)
{
    return !swarm_enterRuntime(NULL,
                               startup, NULL);
}

static swarm_dependency_t dep;
static void startup(void *unused)
{
    unsigned i;
    (void)unused;
    swarm_dependency_init(&dep, COUNT, done, NULL);
    for(i=0; i<COUNT; i++)
        swarm_scheduleGeneral(hello, (void *)(size_t)i);
}

static void hello(void *i)
{
    const unsigned i = (size_t)_i;
    printf("%u: Hello,\n", i);
    swarm_scheduleGeneral(world, _i);
}

static void world(void *i)
{
    const unsigned i = (size_t)_i;
    printf("%u: world!\n", i);
    swarm_satisfy(&dep, 1);
}

static void done(void *unused)
{
    (void)unused;
    puts("All done!");
    swarm_shutdownRuntime(NULL);
}
```
Outline

• Introduction to SWARM
• Programming in SWARM
• **Atomic Operations in SWARM**
• Parallel For Loop in SWARM
Example: Data Race in Computing Sum

Compute $sum = A1 + A2 + A3$

Codelet 0
sum = 0;

Codelet 1
tmp1 = sum;
tmp1 = tmp1 + A1;
sum = tmp1;

Codelet 2
tmp2 = sum;
tmp2 = tmp2 + A2;
sum = tmp2;

Codelet 3
tmp3 = sum;
tmp3 = tmp3 + A3;
sum = tmp3;

Codelet 4
output(sum);
Example: Data Race in Computing Sum

Compute $\text{sum} = A_1 + A_2 + A_3$

Result: $\text{tmp1} = \text{tmp2} = \text{tmp3} = 0$

Executed instructions
Executing instructions
Unexecuted instructions
Example: Data Race in Computing Sum

Compute $\text{sum} = A_1 + A_2 + A_3$

Result: $\text{tmp}_1 = A_1$
$\text{tmp}_2 = A_2$
$\text{tmp}_3 = A_3$

Executed instructions
Executing instructions
Unexecuted instructions

Codelet 0
$\text{sum} = 0$;

Codelet 1
$\text{tmp}_1 = \text{sum}$;  
$\text{tmp}_1 = \text{tmp}_1 + A_1$;  
$\text{sum} = \text{tmp}_1$;

Codelet 2
$\text{tmp}_2 = \text{sum}$;  
$\text{tmp}_2 = \text{tmp}_2 + A_2$;  
$\text{sum} = \text{tmp}_2$;

Codelet 3
$\text{tmp}_3 = \text{sum}$;  
$\text{tmp}_3 = \text{tmp}_3 + A_3$;  
$\text{sum} = \text{tmp}_3$;

Codelet 4
output($\text{sum}$);
Example: Data Race in Computing Sum

Compute \( \text{sum} = A_1 + A_2 + A_3 \)

Result: \( \text{tmp1} = A_1 \)
\( \text{tmp2} = A_2 \)
\( \text{tmp3} = A_3 \)
\( \text{sum} = A_1 \)

Executed instructions
Executing instructions
Unexecuted instructions

Codelet 0
\( \text{sum} = 0; \)

Codelet 1
\( \text{tmp1} = \text{sum}; \)
\( \text{tmp1} = \text{tmp1} + A_1; \)
\( \text{sum} = \text{tmp1}; \)

Codelet 2
\( \text{tmp2} = \text{sum}; \)
\( \text{tmp2} = \text{tmp2} + A_2; \)
\( \text{sum} = \text{tmp2}; \)

Codelet 3
\( \text{tmp3} = \text{sum}; \)
\( \text{tmp3} = \text{tmp3} + A_3; \)
\( \text{sum} = \text{tmp3}; \)

Codelet 4
\( \text{output(sum);} \)
Example: Data Race in Computing Sum

Execute instructions
Executing instructions
Unexecuted instructions

Codelet 0
sum = 0;

Codelet 1

tmp1 = sum;
tmp1 = tmp1 + A1;
sum = tmp1;

Codelet 2

tmp2 = sum;
tmp2 = tmp2 + A2;
sum = tmp2;

Codelet 3

tmp3 = sum;
tmp3 = tmp3 + A3;
sum = tmp3;

Codelet 4
output(sum);

Compute sum = A1 + A2 + A3

Result: tmp1 = A1
tmp2 = A2
tmp3 = A3
sum = A2

Executed instructions
Executing instructions
Unexecuted instructions
Example: Data Race in Computing Sum

Executed instructions
Executing instructions
Unexecuted instructions

Computes sum = A1 + A2 + A3

Result:
- tmp1 = A1
- tmp2 = A2
- tmp3 = A3
- sum = A3

```
// Codelet 0
sum = 0;

// Codelet 1
tmp1 = sum;
tmp1 = tmp1 + A1;
sum = tmp1;

// Codelet 2
tmp2 = sum;
tmp2 = tmp2 + A2;
sum = tmp2;

// Codelet 3
tmp3 = sum;
tmp3 = tmp3 + A3;
sum = tmp3;

// Codelet 4
output(sum);
```
Example: Data Race in Computing Sum

Compute sum = A1 + A2 + A3

Result: output A3
Wrong result!!
Example: Data Race in Computing Sum – Solution

The three instructions must be executed \textit{atomically}.

Codelet $i$

\begin{align*}
\text{tmp1} &= \text{sum}; \\
\text{tmp1} &= \text{tmp1} + \text{A1}; \\
\text{sum} &= \text{tmp1};
\end{align*}

The three instructions must be executed as if one instruction. When the codelet computes sum, the other codelets must not change the value of sum.
Example: Data Race in Computing Sum – Solution

Compute sum = A1 + A2 + A3

Codelet 0
sum = 0;

Codelet 1
atomic
tmp1 = sum;
tmp1 = tmp1 + A1;
sum = tmp1;

Codelet 2
atomic
tmp2 = sum;
tmp2 = tmp2 + A2;
sum = tmp2;

Codelet 3
atomic
tmp3 = sum;
tmp3 = tmp3 + A3;
sum = tmp3;

Codelet 4
output(sum);
Example: Data Race in Computing Sum – Solution

Compute sum = A1 + A2 + A3

Execution process: tmp2 = 0
   tmp2 = A2
   sum = A2

Executed instructions
Executing instructions
Unexecuted instructions

Codelet 0
sum = 0;

Codelet 1 atomic
tmp1 = sum;
tmp1 = tmp1 + A1;
sum = tmp1;

Codelet 2 atomic
tmp2 = sum;
tmp2 = tmp2 + A2;
sum = tmp2;

Codelet 3 atomic
tmp3 = sum;
tmp3 = tmp3 + A3;
sum = tmp3;

Codelet 4
output(sum);
Example: Data Race in Computing Sum – Solution

Compute sum = A1 + A2 + A3
Execution process: tmp3 = A2
   tmp3 = A2 + A3
   sum = A2 + A3

Executed instructions
Executing instructions
Unexecuted instructions

Codelet 0
sum = 0;

Codelet 1
atomic
tmp1 = sum;
tmp1 = tmp1 + A1;
sum = tmp1;

Codelet 2
atomic
tmp2 = sum;
tmp2 = tmp2 + A2;
sum = tmp2;

Codelet 3
atomic
tmp3 = sum;
tmp3 = tmp3 + A3;
sum = tmp3;

Codelet 4
output(sum);
Example: Data Race in Computing Sum – Solution

Compute \( \text{sum} = A_1 + A_2 + A_3 \)

Execution process:
- \( \text{tmp1} = A_2 + A_3 \)
- \( \text{tmp1} = A_1 + A_2 + A_3 \)
- \( \text{sum} = A_1 + A_2 + A_3 \)

Executed instructions
- Executing instructions
- Unexecuted instructions

```
Codelet 0
sum = 0;

Codelet 1
atomic
\( \text{tmp1} = \text{sum}; \)
\( \text{tmp1} = \text{tmp1} + A_1; \)
\( \text{sum} = \text{tmp1}; \)

Codelet 2
atomic
\( \text{tmp2} = \text{sum}; \)
\( \text{tmp2} = \text{tmp2} + A_2; \)
\( \text{sum} = \text{tmp2}; \)

Codelet 3
atomic
\( \text{tmp3} = \text{sum}; \)
\( \text{tmp3} = \text{tmp3} + A_3; \)
\( \text{sum} = \text{tmp3}; \)

Codelet 4
output(sum);
```
Example: Data Race in Computing Sum – Solution

Compute sum = A1 + A2 + A3

Result: output A1 + A2 + A3

Executed instructions
Executing instructions
Unexecuted instructions

Codelet 0
sum = 0;

Codelet 1
atomic
tmp1 = sum;
tmp1 = tmp1 + A1;
sum = tmp1;

Codelet 2
atomic
tmp2 = sum;
tmp2 = tmp2 + A2;
sum = tmp2;

Codelet 3
atomic
tmp3 = sum;
tmp3 = tmp3 + A3;
sum = tmp3;

Codelet 4
output(sum);
Atomic Operations in SWARM

• `swarm_atomic_getAndAdd(var, val)`
  – Atomically do the following work
    • `ret = var`
    • `var = var + val`
    • `return ret`

• `swarm_atomic_cmpAndSet(var, val, newVal)`
  – Atomically do the following work
    • If `var == val`, then `{var = newVal; return true;}`
    • Otherwise, return `false`

- General information about atomic operations: share/doc/swarm/programmers-guide/sec_coding_atomics.htm and share/doc/swarm/programmers-guide/sec_coding_atomics_naming.htm
- Information about atomic get and add: share/doc/swarm/programmers-guide/sec_coding_atomics_rmw.htm
- Information about atomic compare and set: share/doc/swarm/programmers-guide/sec_coding_atomics_access.htm
Outline

• Introduction to SWARM
• Programming in SWARM
• Atomic Operations in SWARM
• Parallel For Loop in SWARM
Problem formulation: Suppose we have the following for loop where loop iterations can be executed in arbitrary order. How can we parallel the for loop in SWARM?

\[
\text{for (i = 0; i < N; i++)}
\text{ foo(i);}
\]
Problem formulation: Suppose we have the following for loop where loop iterations can be executed in arbitrary order. How can we parallel the for loop in SWARM?

```plaintext
for (i = 0; i < N; i++)
    foo(i);
```

Methodology 1: Spawn N codelets. Each codelet does one foo(i). Not recommended due to heavy overhead.
Parallel For Loop in SWARM (3)

Problem formulation: Suppose we have the following for loop where loop iterations can be executed in arbitrary order. How can we parallel the for loop in SWARM?

```c
for (i = 0; i < N; i++)
    foo(i);
```

Methodology 1: Spawn N codelets. Each codelet does one foo(i). Not recommended due to heavy overhead.

Methodology 2: Spawn k codelets. Each codelet does N/k foo(i)s. Good for balanced workload. Not good for unbalanced workload.
Problem formulation: Suppose we have the following for loop where loop iterations can be executed in arbitrary order. How can we parallel the for loop in SWARM?

\[
\text{for } (i = 0; i < N; i++) \\
\quad \text{foo}(i);
\]

Methodology 1: Spawn N codelets. Each codelet does one foo(i). Not recommended due to heavy overhead.

Methodology 2: Spawn k codelets. Each codelet does N/k foo(i)s. Good for balanced workload. Not good for unbalanced workload.

Methodology 3: Spawn k codelets. Each codelet dynamically execute foo(i)s. Good for unbalanced workload.
Parallel For Loop in SWARM (5)

static void startup(void *unused)
{
    unsigned i;
    (void)unused;
    // COUNT is total number of threads
    swarm_dependency_init(&dep, COUNT, done, NULL);
    for(i=0; i<COUNT; i++)
        swarm_scheduleGeneral(dotproduct, (void *)(size_t)i);
}

static void dotproduct(void *tid)
{
    const unsigned tid = (size_t)tid;
    unsigned i;
    // LEN is length of the array
    sum[tid] = 0;
    for (i = tid * LEN / COUNT; i < (tid + 1) * LEN / COUNT; i++)
        sum[tid] += v1[i] * v2[i];

    swarm_satisfy(&dep, 1);
}

static void done(void *unused)
{
    unsigned i;
    (void)unused;
    int result;

    result = 0;
    for (i = 0; i < COUNT; i++)
        result += sum[i];

    printf("Result is : %d\n", result);
    swarm_shutdownRuntime(NULL);
}

Example of using methodology 2 for vector dot product
Parallel For Loop in SWARM (6)

```
for (i = 0; i < N; i++)
  foo(i);
```

Methodology 3: Spawn $k$ codelets. Each codelet dynamically execute $\text{foo}(i)$s. How?

**Codelet**

1. Get first index of unexecuted loop iteration and stored in $i$
2. Increase the index by $\text{CHUNK\_SIZE}$
3. Executes $\text{foo}(i), \text{foo}(i+1), \ldots, \text{foo}(i+\text{CHUNK\_SIZE}-1)$

Hints: Steps (1) and (2) must be done atomically.
- Once $i \geq N$, the codelet is completed
- Correctly handle the case that $i + \text{CHUNK\_SIZE} \geq N$
Set and Get maximum number of threads

**Set maximum number of threads**

```c
swarm_Runtime_params_t p;
swarm_Runtime_params_init(&p);
if (M_NUM_THREADS > 0) p.maxThreadCount = m_numthreads;

if(!swarm_enterRuntime(&p, startup, _ctxt)) { //startup : entry codelet
    //_ctxt: parameter passing to startup
    fprintf(stderr, "%s: unable to start SWARM runtime\n", *argv);
    return 64;
}
```
Parallel For Loop in SWARM (8)

Set and Get maximum number of threads

Get maximum number of threads

```c
unsigned GetSwarmThreadCount() {
    // return maximum number of threads
    const swarm_ThreadLocale_t *top = swarm_topmostLocale;
    size_t k;
    if(!top) return 1;
    k = swarm_Locale_getChildren(
        swarm_ThreadLocale_to_Locale(top), NULL, 0);
    return k+!k;
}
```