Dataflow Models: Present and Future

CPEG 852 - Spring 2014
Advanced Topics in Computing Systems

Guang R. Gao
ACM Fellow and IEEE Fellow
Endowed Distinguished Professor
Electrical & Computer Engineering
University of Delaware
ggao.capsl@gmail.com
Outline

• Introduction
• Evolution of the **Codelet** Execution Model
• Del-CAM - The Delaware Codelet Abstract Machine
• A Remark on Runtime Software Technology vs. OS
• DARTS — *Delaware Adaptive Runtime System*
• SWARM — *SWift Adaptive Runtime Machine*
• Summary
Issues and Challenges
DOE X-Stack Objectives

• **Scalability**: Enables applications to strongly scale to Exascale levels of parallelism;

• **Programmability**: Clearly reduces the burden we are placing on high performance programmers;

• **Performance Portability**: Eliminates or significantly minimizes requirements for porting to future platforms;

• **Resilience**: Properly manages fault detection and recovery at all components of the software stack; and

• **Energy Efficiency**: Maximally exploits dynamic energy saving opportunities, leveraging the tradeoffs between energy efficiency, resilience, and performance.
DOE X-Stack Awardees

**Codelet based**

**Traleika Glacier** (S. Borkar)
http://sites.google.com/site/traleikaglacierxstack/

**DynAX** (G. Gao)
http://www.etinternational.com/xstack

**GVR** (A. Chien)
http://gvr.cs.uchicago.edu

**X PRESS** (R. Brightwell)
http://xstack.sandia.gov/xpress

**D-TEC** (D. Quinlan, S. Amarasinghe)
http://www.dtec-xstack.org

**XTUN E** (M. Hall)
http://ctop.cs.utah.edu/x-tune/

**SLEE C** (M. Kulkarni)
https://engineering.purdue.edu/~milind/sleec/

**DEGAS** (K. Yelick)
http://crd.lbl.gov/groups-depts/future-technologies-group/projects/DEGAS/

**CORVETTE** (K. Sen)
http://crd.lbl.gov/groups-depts/future-technologies-group/projects/corvette/
Straw-man Architecture for the Traleika Glacier (X-Stack) project

Control Engine (CE)
- 32KB I$ RF?
- 64KB SP GP Int
System SW

Block (8 XE + CE)
- XE XE XE XE
- XE XE XE
- 1MB shared L2

Execution Engine (XE)
- 32KB I$ RF?
- 64KB SP DP FP FMAC
Application specific
Event driven tasks (EDT):
Dataflow inspired, tiny codelets (self contained).
Non blocking, no preemption.

Programming model:
Separation of concerns: Domain specification & HW mapping.
Express data locality with hierarchical tiling.
Global, shared, non-coherent address space.
Optimization and auto generation of EDTs (HW specific).

Execution model:
Dynamic, event-driven scheduling, non-blocking.
Dynamic decision to move computation to data.
Observation based adaption (self-awareness).
Implemented in the runtime environment.

Separation of concerns:
User application, control, and resource management.
Outline

• Introduction
• Evolution of the **Codelet** Execution Model
• Del-CAM - The Delaware Codelet Abstract Machine
• A Remark on Runtime Software Technology vs. OS
• DARTS – *Delaware Adaptive Runtime System*
• SWARM – *SWift Adaptive Runtime Machine*
• Summary
A Quiz: Have you heard the following terms?

Actors (dataflow) ?

strand ?

fiber ?

codelet ?
Coarse-Grain vs. Fine-Grain Multithreading

Coarse-Grain thread-
The family home model

Fine-Grain *non-preemptive* thread-
The “hotel” model

The “hotel” model

Thread Unit
Executor Locus
A Single Thread

Thread Unit
A Thread Pool

CPU
Memory

CPU
Memory

CPU
Memory
A version of this slide was presented in my invited talk at Fran Allen’s retirement party July 2002

Coarse-Grain vs. Fine-Grain Multithreading
What is a Codelet?

• Intuitively:

A unit of computation which interacts with the global state only at its entrance and exit points

• Terminology

- I would be very cautious to use the term “functional programming” here – which usually means “stateless”!

- But, I like the term of single-assignment and dataflow programming models
Codelet Graph

• A Codelet Graph $G = \{V,E\}$ is a Graph where:
  – $G$ is a set of nodes, where each corresponds to a Codelet.
  – $E$ is a set of directed edges, each connecting a pair of Codelet nodes (except in/out edges)

NOTE: I did not restrict the semantics of edges only on dependencies.
Operational Semantics of Codelets

Enabling/Firing Rules

Consider a Codelet graph G – with an assignment of events on some of its edges:

• A codelet is *enabled* if
  – An event is present on each of its input edges;
  – none of the output edges may have any events.

• An enabled event can be scheduled for execution (i.e. *fired*). The firing of a Codelet will remove all input events (one from each input), and will produce output events, one on each output.
The Codelet: A Fine-Grain Piece of Computing

- A Codelet is fired when all its inputs are available.
- Inputs can be data or resource conditions.
- Fundamental properties of Data-Flow: **Determinacy**, **Repeatability**, **Composability**, among others.

This Looks Like Data Flow!  
- *Jack Dennis*
Evolution of Multithreaded

A version of this slide was presented in my invited talk at Fran Allen’s retirement party July 2002

- CHoPP’77 → CHoPP’87
- MASA → Alwife
  - Halstead → Agarwal 1989-96
  - B. Smith → B. Smith
- CDC 6600 → HEP
  - 1964 → B. Smith
- HEP → Tera
  - B. Smith → Tera
  - B. Smith → Eldorado
  - Gurd & Watson 1982
- Non-dataflow based
- Dataflow model inspired
  - MIT TTDA 1980
  - Manchester 1982
  - Arg-Fetching Dataflow 1987-88
  - Dennis 1972
  - MIT
  - Dennis Gao 1987-88
  - MDFA 1989-93
- Static Dataflow
  - Dennis 1972
  - MIT
  - Gao et al. 1993-2006
- EARTH
  - Gao et al. (Hum, Theobald, etc.) 1993-2006
- HTVM/TNT-X
  - Gao et al. (DelCuvillo - TNT) 2000-2010
- Codelet Model
  - Gao et al. 2009-
More Comments on Codelets model

• Codelets execution is intended non-preemptive.

• So we should ensure that between any two statements in a Codelet there should not be any dependences that will take long latency to resolve (or satisfied).

QUIZ: how to determine this?

• History Remarks: Ph.D theses under Gao’s supervision during EARTH Project in 1990s – [Hum 92, Newmarwarkar 97, Tang 98, Theobald 99] and others.
## Recent Ph.D Theses Completed at CAPSL

<table>
<thead>
<tr>
<th>Author</th>
<th>Title</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Juergen Ributzka</td>
<td><em>Concurrency and Synchronization in the Modern Many-Core Era: Challenges and Opportunities</em></td>
<td>Spring 2013</td>
</tr>
<tr>
<td>Daniel Orozco</td>
<td><em>TIDeFlow: A Dataflow-inspired execution model for high performance computing programs</em></td>
<td>Spring 2012</td>
</tr>
<tr>
<td>Joseph B. Manzano</td>
<td><em>A comparison between virtual code management techniques</em></td>
<td>Summer 2011</td>
</tr>
<tr>
<td>Ge Gan</td>
<td><em>Programming model and execution model for OpenMP on the Cyclops-64 manycore processor</em></td>
<td>Spring 2010</td>
</tr>
<tr>
<td>Long Chen</td>
<td><em>Exploring novel many-core architectures for scientific computing</em></td>
<td>Fall 2010</td>
</tr>
<tr>
<td>Fei Chen</td>
<td><em>Enabling system validation for the many-core supercomputer</em></td>
<td>Summer 2009</td>
</tr>
<tr>
<td>Juan del Cuvillo</td>
<td><em>Breaking away from the OS shadow: A program execution model aware thread virtual machine for multicore architectures</em></td>
<td>Summer 2008</td>
</tr>
<tr>
<td>Yuan Zhang</td>
<td><em>Static analyses and optimizations for parallel programs with synchronization</em></td>
<td>Summer 2008</td>
</tr>
<tr>
<td>Weirong Zhu</td>
<td><em>Efficient synchronization for a large-scale multi-core chip architecture</em></td>
<td>Spring 2007</td>
</tr>
</tbody>
</table>
Outline

• Introduction
• Evolution of the Codelet Execution Model
• Del-CAM - The Delaware Codelet Abstract Machine
• A Remark on Runtime Software Technology vs. OS
• DARTS – Delaware Adaptive Runtime System
• SWARM – SWift Adaptive Runtime Machine
• Summary
Challenges and Opportunity of Designing An Abstract Machine?

• What is an Abstract Machine?
• What is the motivation of a “good” abstract machine?
• What is the Del-CAM - Delaware Codelet Abstract Machine?
The Challenges

Performance, Programmability and Productivity

“Humans are quickly overwhelmed by concurrency and find it much more difficult to reason about concurrent than sequential code. Even careful people miss possible inter-leavings among even simple collections of partially ordered operations”.

Sutter and Larus
Impact on Abstract Machine Models

The “PPPP” Problem

Performance, Productivity, Portability, Power

Performance

Effort + Expert Knowledge

Best Performance

Good Performance

A1

A2

A3
Observation

Not All Abstract Architecture Models Are Equal!
Execution Model, Abstract Machines, and **Runtime Software**
High-Level Programming API (SHMeM, etc.)

Software packages
Program libraries
Utility applications

Compilers
Tools/SDK

Users

Programming Models/Environment

TNT API

TNT Execution Model

TNT Abstract Machine

TNT Runtime System

Hardware Architecture

IBM Cyclops64: A Case Study
The Delaware Codelet Abstract Machine (Del-CAM) has its roots in:

- The Dataflow Machine Models in 1970s [e.g. J.B. Dennis, 1972, and DennisGao88, see the history landscape on slide 20].
- The EARTH (Efficient Architecture for Running Threads) [e.g. Hum, et. Al., 1994, Theobald, 1999, etc.].
- The Delaware Codelet Execution Model.
Outline

• Introduction
• Evolution of the Codelet Execution Model
• Del-CAM - The Delaware Codelet Abstract Machine
• A Remark on Runtime Software Technology vs. OS
• DARTS – Delaware Adaptive Runtime System
• SWARM – Swift Adaptive Runtime Machine
• Summary
A Traditional View of Computer Systems

[Patterson & Silberschatz]
What is OS Anyway?

The operating system acts as a host for computing applications ... one of the purposes of an operating system is to handle the details of the operation of the hardware. ....

Example: virtual memory support

[From Wikipedia]
What is OS Anyway? (cont’d)

Operating systems offer a number of services to application programs and users. Applications access these services through \textit{application programming interfaces} (APIs) or \textit{system calls}. Example: PRINT command, etc.

[From Wikipedia]
Traditional OS Functions

- Process Management & Services (e.g. CPU Scheduling)
- Register allocation
- Memory Management & Services (e.g. Virtual Memory)
- Instruction Scheduling
- I/O Management & Services (e.g. Device Drivers)
- Branch Prediction
- Protection & Security Services
- Control Speculation
- File Systems

Which services / functions a traditional OS has?

Which services / functions do not belong to traditional OS?
How About OS in Many-Core Era?
OS in the Many-Core Era

Which services / functions a traditional OS has?

Runtime System

Process Management & Services (e.g. Process Scheduling)

Codelet Scheduling

Memory Management & Services (e.g. Virtual Memory)

Fine Grain Memory Management (e.g. Data Percolation)

Protection & Security Services

Energy Management (Coarse Grain)

Energy Management (Fine Grain)

File Systems

Locality Management

But where can we buy a Runtime System for Many-Core Architectures?

Which services/ functions do not belong to traditional OS?
Outline

• Introduction
• Evolution of the **Codelet** Execution Model
• Del-CAM - The Delaware Codelet Abstract Machine
• A Remark on Runtime Software Technology vs. OS
• DARTS – *Delaware Adaptive Runtime System*
• SWARM – *SWift Adaptive Runtime Machine*
• Summary
Codelet Model: a Short Introduction

• A codelet is a sequence of machine instructions that run non-preemptively

• **Codelet firing rule**: a codelet can only start executing when all of the events it depends on are satisfied.

• Events can be related to:
  – Data (*i.e.* the input data is ready to be used)
  – Other resources, such as guaranteed minimal bandwidth; the use of an accelerator (*e.g.*, Xeon Phi or GPU); minimal frequency; *etc.*
Codelet Model: the Codelet Abstract Machine (AMM)

- Codelets are assumed to be run on a deeply hierarchical and distributed many-core machine.
- A codelet machine exposes two types of cores: Synchronization Units (SUs), and Computation Units (CUs).
- A codelet AMM groups some memory, at least 1 SU, and one or more CUs into a cluster. The codelet AMM features several of these clusters.
Codelet Model: Codelet Graph (CDG)

• Once a codelet has processed a piece of data, it signals its availability to the codelets that depend on it.
• Signaling is performed as codelets execute, not just at the end of processing.
• Codelets are linked together by their data dependencies.
Codelet Model: Threaded Procedures

- Codelets are pieces of code that process their input data – but we haven’t discussed where the data is stored yet!
- Portions of the overall codelet graph are grouped together, along with the data they need to read from and write to.
- Such containers are called Threaded Procedures (or TPs). They are invoked in a control-flow manner, much like a regular function.
Codelet Model: Loops

• Loops are containers of a codelet graph, with an additional back-edge.

• Loops can be defined at various granularities:
  – Across several TPs – all iterations will then become TP invocations, and dynamically allocate new codelets
  – Within a given TP, across several codelets: they will persist over time until the loop reaches its end.
DARTS: the Delaware Adaptive Run-Time System

• DARTS is a *faithful* implementation of the Codelet Model:
  – It reproduces the codelet abstract machine, including compute units and synchronization units
  – it explicitly creates two levels of parallelism (TPs and codelets)

• It is built with modularity in mind:
  – It has an object-oriented design and is written in C++
  – It is meant to be a research vehicle for further developments of the codelet model.
  – Hence, it is easily extensible
class Codelet { 
private:
    SyncSlot slot_; 
    uint64_t metadata;
protected:
    ThreadedProcedure* myTP_; 
public:
    Codelet(uint32_t deps, uint32_t reset, ThreadedProcedure* frame, uint64_t meta) : slot_(deps, reset), metadata(meta), myTP_(frame) {} 
    virtual void fire(void) = 0; 
    void decDep(); 
}

class ThreadedProcedure { 
protected:
    // garbage collection data
public:
    ThreadedProcedure* parentTP; 
    ThreadedProcedure(void); 
    virtual ~ThreadedProcedure(void); 
    // Skipping garbage collection 
    // method declarations...
    void add(Codelet* toAdd); 
}
DARTS: Codelets and Threaded Procedures (2)

```cpp
struct SomeTP : public ThreadedProcedure {
    // DATA
    int *input,
        a, b, c, /* local to CDG */
        *output;
    // CODE
    CodeletA A; CodeletB B;
    CodeletC C; CodeletD D;
    Codelet* toSignal;
    SomeTP(int* in, int* out, Codelet* toSig) :
        input(in), a(0), b(0), c(0), output(out),
        A(this), B(this), C(this), D(this), toSignal(toSig)
    {
        add(&A);
    }
};
```
DARTS: Codelets and Threaded Procedures (3)

```cpp
struct CodeletA : public Codelet {
    CodeletA(ThreadedProcedure* frame) : Codelet(0,0,frame,SHORTWAIT) {} 
    virtual void fire(void);
}

void CodeletA::fire(void) {
    SomeTP* frame = (SomeTP*) myTP_;  
    // Do something with the TP frame...  
    frame->B.decDep();  
    frame->C.decDep();
}
```
DARTS: Loops

• In DARTS, loops are essentially TPs that create a “loop schema” following a “parallel for” pattern.

• Subtleties:
  • Codelets stay allocated for the loop’s duration.
  • The “loop control codelet” tasked resets the codelets dependencies for the next iteration.
DARTS: Fibonacci Example
struct Checker
  : public Codelet
{
  virtual void fire (void);
Checker(ThreadedProcedure* frame)
  : Codelet(0,0,frame,SHORTWAIT)
{}
}

struct Adder
  : public Codelet
{
  virtual void fire (void);
Checker(ThreadedProcedure* frame)
  : Codelet(2,2,frame,LONGWAIT)
{}
DARTS: Fibonacci Example (2)

```c++
void Checker::fire(void) {
    Fib* frame = static_cast<Fib*>(myTP_);
    int n = frame->num, *r = frame->result;
    if (n < 2) { *r = n; frame->final->decDep(); return; }
    invoke<Fib>(frame, n-1, &frame->n1, &frame->adder);
    invoke<Fib>(frame, n-2, &frame->n2, &frame->adder);
}

void Adder::fire(void) {
    Fib* frame = static_cast<Fib*>(myTP_);
    *frame->result = frame->n1 + frame->n2;
    frame->final->decDep();
}
```
DARTS: Experimental Results

- All experiments run on a 4 × 12 core machine
- 1 FPU shared between 2 cores
- Frequency: 2.4GHz
- 16KB L1 data cache (private)
- 2MB L2 unified cache (shared by 2 cores)
- 6MB L3 unified cache (shared by 6 cores)
- 128GB DDR3 DRAM
DARTS: Experiments with DGEMM

- DGEMM: Double General Matrix Multiplication
- Used AMD’s Core Math Library (ACML) as sequential building block for DARTS
- Compared several scheduling policies with parallel ACML.
DARTS: Experiments with DGEMM

Weak scaling, 48-core case (i.e., only the inputs vary)
DARTS: Experiments with DGEMM

Strong scaling, 1000 × 1000 matrices (i.e., only # of cores vary)
DARTS: Experiments with DGEMM

Strong scaling, $10000 \times 10000$ matrices (i.e., only # of cores vary)
DARTS: Experiments with Graph500

- Only modified the breadth-first search (BFS) phase
- Compared with reference OpenMP parallelization
DARTS: Experiments with Graph500

Weak scaling, 48-core case (i.e., only the inputs vary)

Unit: Traversed Edges Per Second (TEPS)
DARTS: Experiments with Graph500

Strong scaling, input = $2^{18}$ (i.e., only the inputs vary)

Unit: Traversed Edges Per Second (TEPS)
Outline

• Introduction
• Evolution of the Codelet Execution Model
• Del-CAM - The Delaware Codelet Abstract Machine
• A Remark on Runtime Software Technology vs. OS
• DART – Delaware Adaptive Runtime Testbed
• SWARM – SWift Adaptive Runtime Machine
• Summary
High-Level Programming API
(MPI, Open MP, CnC, Xio, Chapel, etc.)

Software packages
Program libraries
Utility applications

Compilers
Tools/SDK

SWARM API

SWARM Runtime System
Hardware Architecture

SWARM Abstract Machine

Codelet Execution Model and Abstract Machine

Codelet Execution Model

Programming Models/Environment
SWARM Execution Overview

Enabled Tasks

Tasks enabled

Tasks mapped to resources

SWARM

Tasks with Unsatisfied Dependencies

Dependences satisfied

Available Resources

Resources allocated

Resources released

Resources in Use
SWARM Technology: Differentiation Features

- Fine-grain asynchrony.
- Dynamic dependency exploitation (beyond static DAG parallelism).
- Dynamic load-balancing and resource management.
- Event-driven, self-aware, adaptive control of energy and resiliency.
- Determinacy and repeatability – key to fault tolerance and recovery.
Difference Between OpenMP and SWARM

OpenMP
Coarse-grain execution model

- Master thread
- Slave threads

Sequential task
- Wake up all threads
- Parallel tasks
- Barrier
- Sequential task
- Wake up all threads
- Barrier

SWARM
Fine-grain execution model
- tasks (codelets)
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 `swarm_satisfy()` to decrease the value of any dependency counter.
- SWARM runtime handles codelet schedule and dependency maintenance.
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.

- **Thread is executing codelet1**
  - **Codelet2**
    - dep2 = 0
    - Enabled
  - **Codelet1**
    - dep1 = 0
    - Firing
    - Call satisfy() to decrease dep3 by 1
  - **Codelet3**
    - dep3 = 2
    - Dormant
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.

Thread is free

Codelet1

\[ \text{dep1} = 0 \]

\[ \text{dep2} = 0 \]

\[ \text{dep3} = 1 \]

**Done**

Call `satisfy()` to decrease dep3 by 1

**Enabled**

**Dormant**

**Codelet1**

**Codelet2**

**Codelet3**
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.

Thread is free

- **Codelet1**
  - Dep1 = 0
  - Call `satisfy()` to decrease dep3 by 1
  - Done

- **Codelet2**
  - Dep2 = 0
  - Call `satisfy()` to decrease dep3 by 1
  - Done

- **Codelet3**
  - Dep3 = 0
  - Enabled
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.

Thread is free

dep1 = 0

Codelet1
Call satisfy() to decrease dep3 by 1

dep2 = 0

Codelet2
Call satisfy() to decrease dep3 by 1

dep3 = 0

Codelet3

Done

Done

Done
Example: Hello World

Codelet graph of hello world
Example: Hello World

```c
#include <stdio.h>
#include <eti/swarm_convenience.h>

#define COUNT 8
swarm_Dep_t dep;

CODELET_DEL(startup);
CODELET_DEL(hello);
CODELET_DEL(world);
CODELET_DEL(done);
int main(void)
{
    return swarm_posix_enterRuntime(NULL, &CODELET(startup), NULL, NULL);
}

CODELET_IMPL_BEGIN_NOCANCEL(startup)
    swarm_Dep_init(&dep, COUNT, &CODELET(done), NULL, NULL);
    int i;
    for (i=0; i<COUNT; i++)
        swarm_schedule(&CODELET(hello), (void*) i, NULL, NULL);
CODELET_IMPL_END;

CODELET_IMPL_BEGIN_NOCANCEL(hello)
    printf("%d: Hello, \n", (int)THIS);
    swarm_schedule(&CODELET(world) THIS, NULL, NULL, NULL);
    CODELET_IMPL_END;

CODELET_IMPL_BEGIN_NOCANCEL(world)
    printf("%d: world!\n", (int) THIS);
    swarm_Dep_satisfy(&dep, 1);
    CODELET_IMPL_END;

CODELET_IMPL_BEGIN_NOCANCEL(done)
    printf("Done\n");
    swarm_shutdownRuntime(NULL);
    CODELET_IMPL_END;
```

8 in all

...
Example: Hello World

```c
#include <stdio.h>
#include <eti/swarm_convenience.h>

#define COUNT 8
swarm_Dep_t dep;

CODELET_DEL(startup);
CODELET_DEL(hello);
CODELET_DEL(world);
CODELET_DEL(done);

int main(void)
{
    return swarm_posix_enterRuntime(NULL,
&CODELET(startup), NULL, NULL);
}

CODELET_IMPL_BEGIN_NOCANCEL(startup)
    swarm_Dep_init(&dep, COUNT, &CODELET(done),
NULL, NULL);
    int i;
    for (i=0; i<COUNT; i++)
        swarm_schedule(&CODELET(hello), (void*) i, NULL,
NULL, NULL);
CODELET_IMPL_END;

8 in all

......

CODELET_IMPL_BEGIN_NOCANCEL(hello)
    printf("%d: Hello, \n", (int)THIS);
    swarm_schedule(&CODELET(world) THIS, NULL, NULL, NULL);
CODELET_IMPL_END;

CODELET_IMPL_BEGIN_NOCANCEL(world)
    printf("%d: world!\n", (int)THIS);
    swarm_Dep_satisfy(&dep, 1);
CODELET_IMPL_END;

CODELET_IMPL_BEGIN_NOCANCEL(done)
    printf("Done\n");
    swarm_shutdownRuntime(NULL);
CODELET_IMPL_END;
```
Example: Hello World

```c
#include <stdio.h>
#include <eti/swarm_convenience.h>

#define COUNT 8
swarm_Dep_t dep;

CODELET_DEL(startup);
CODELET_DEL(hello);
CODELET_DEL(world);
CODELET_DEL(done);
int main(void)
{
    return swarm_posix_enterRuntime(NULL, &CODELET(startup), NULL, NULL);
}

```
#include <stdio.h>
#include <eti/swarm_convenience.h>

#define COUNT 8
swarm_Dep_t dep;

CODELET_DEL(startup);
CODELET_DEL(hello);
CODELET_DEL(world);
CODELET_DEL(done);

int main(void)
{
    return swarm_posix_enterRuntime(NULL, 
    &CODELET(startup), NULL, NULL);
}

CODELET_IMPL_BEGIN_NOCANCEL(startup)
    swarm_Dep_init(&dep, COUNT, 
    &CODELET(done), NULL, NULL);
    int i;
    for (i=0; i<COUNT; i++)
        swarm_schedule(&CODELET(hello), (void*) i, NULL, 
        NULL, NULL);
CODELET_IMPL_END;

8 in all

CODELET_IMPL_BEGIN_NOCANCEL(hello)
    printf(“%d: Hello, \n”, (int) THIS);
    swarm_schedule(&CODELET(world) 
    THIS, NULL, NULL, NULL);
CODELET_IMPL_END;

CODELET_IMPL_BEGIN_NOCANCEL(world)
    printf(“%d: world!\n”, (int) THIS);
    swarm_Dep_satisfy(&dep, 1);
CODELET_IMPL_END;

CODELET_IMPL_BEGIN_NOCANCEL(done)
    printf(“Done\n”);
    swarm_shutdownRuntime(NULL);
CODELET_IMPL_END;
#include <stdio.h>
#include <eti/swarm_convenience.h>

#define COUNT 8
swarm_Dep_t dep;

CODELET_DEL(startup);
CODELET_DEL(hello);
CODELET_DEL(world);
CODELET_DEL(done);

int main(void)
{
    return swarm_posix_enterRuntime(NULL, 
    &CODELET(startup), NULL, NULL);
}

CODELET_IMPL_BEGIN_NOCANCEL(startup)
    swarm_Dep_init(&dep, COUNT, &CODELET(done), 
    NULL, NULL);
    int i;
    for (i=0; i<COUNT; i++)
        swarm_schedule(&CODELET(hello), (void*) i, NULL, 
        NULL, NULL);
CODELET_IMPL_END;

8 in all
SWARM: Case Studies

• *Scientific Kernels*:
  – Cholesky Decomposition.

• *Graph Analytics in BigData*:
  – K-Clique Problem.
Cholesky: OpenMP vs. SWARM

SWARM outperforms on big linear algebra.
Characteristics of CHK in OpenMP

• Static fork-join representation:
  – Imbalance of the code.
  – Dynamic varying demand on runtime resource.

• Coarse-Grain/Barrier Synchronization
Tuned OpenMP vs SWARM

Intel® Xeon Phi™ coprocessor: 240 Threads
CHK on Xeon-Phi: SWARM vs. OMP

SWARM vs OpenMP (New and Old Compiler)

New Compiler/Runtime
- Host OS: Linux
- OS Version: 2.6.32-220.el6.x86_64
- Driver Version: 3653-8
- MPSS Version: 2.1.3653-8
- Host Physical Memory: 65904 MB
- CPU Family: Genuine Intel Family 6 Model 44 Stepping 2
- CPU Speed: 1596 Threads per Core: 2
16 node cluster: Intel Xeon E5-2670 16-core 2.6GHz

SWARM outperforms on big linear algebra
SWARM vs. ScaLAPACK Scalability on XEON Clusters

- At 2 nodes: 86% vs. 70%
- At 4 nodes: 85% vs. 65%
- At 8 nodes: 83% vs. 58%
- At 16 nodes: 82% vs. 48%
- At 32 nodes: 79% vs. 46%
- At 64 nodes: 69% vs. 40%

Respect to a 100% Peak Performance
Characteristics of K-Clique in Hadoop/MR-MPI

• An important and popular bigdata benchmark programs.

• K-C in MR-MPI: Barrier sync, heavy disk usage.

• K-C in MR-SWARM - Asynchronous execution, fine-grain sync and heavy, irregular network usage.
A Hadoop Job

MapReduce 2
Yet Another Resource Negotiator (YARN)

A HAMR Job

SWARM
Data Flow Engine for Flow Control

Hadoop Chain
Batch Processing

HAMR Chain
Realtime Processing
K-cliques: SWARM vs. MR-MPI

SWARM outperforms on big data problems

16 node cluster: Intel Xeon E5-2670 16-core 2.6GHz
More information about SWARM

• SWARM Main page
  – http://www.etinternational.com/swarm

• SWARM Tutorial Wiki
  – http://www.etinternational.com/swarm_wiki

• SWARM Programmer’s Guide

• SWARM API Reference
  – http://www.etinternational.com/downloads/swarm_docs/swarm/api