CPEG 852
Advanced Topics in Computing Systems
The Codelet Model
Specification and Implementation

Stéphane Zuckerman

Computer Architecture & Parallel Systems Laboratory
Electrical & Computer Engineering Dept.
University of Delaware
140 Evans Hall Newark,DE 19716, United States
szuckerm@udel.edu

November 3, 2015

Outline

Outline

1. **The Codelet Model**

Objectives

- Fine-grain parallelism
- Scalable
- Expose maximal parallelism
- Limits non-determinism (determinate-by-default)
- Handles dynamic events (power, resiliency, resource constraints in general)

Definition

A codelet is a sequence of machine instructions which act as an atomically-scheduled unit of computation.

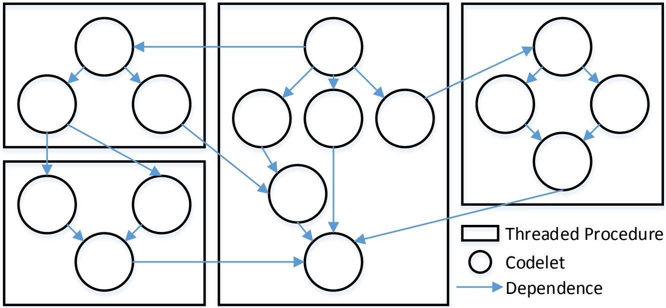
See DARPA-BAA-10-37 2010-2012; Carter et al. 2013; Department of Energy 2012–2014

The Codelet Model

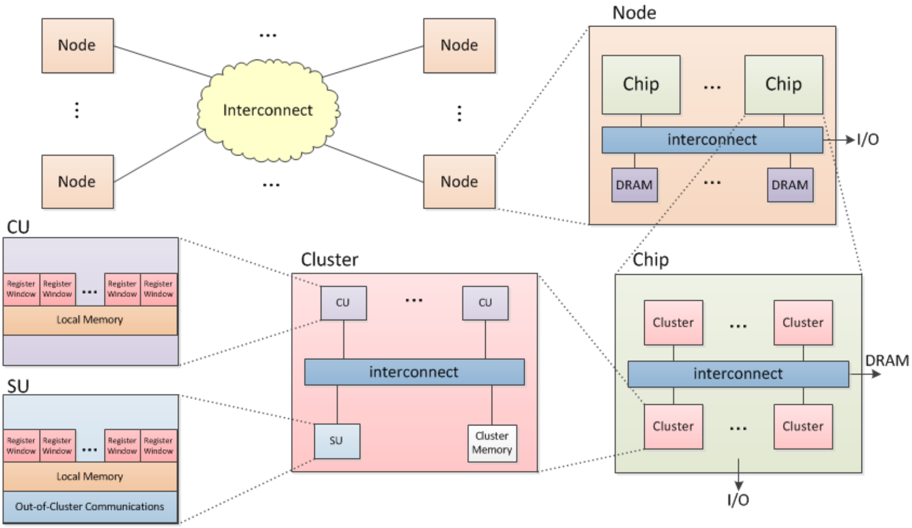
Harnessing Parallelism in Shared-Memory Multi/Many Core Systems

Properties

- Event-driven (availability of data and resources)
- Communicates only through its inputs and outputs
- Non-preemptive (with very *specific* exceptions)
- Requires all data and code to be "local"



The Codelet Abstract Machine



See Stéphane Zuckerman, Suetterlein, et al. 2011

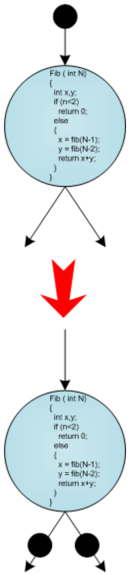
Codelet Graphs: Operational Semantics

Codelet Firing Rule

- Codelet actors are *enabled* once tokens are on each input arc
- Codelet actors fire by
  - consuming tokens
  - performing the operations within the codelet
  - producing a token on each of its output arcs

States of a Codelet

- Dormant: Not all tokens are available
- Enabled: All *data* tokens are available
- Ready: All tokens are available
- Active: The codelet is executing internal operations



See Stéphane Zuckerman, Suetterlein, et al. 2011

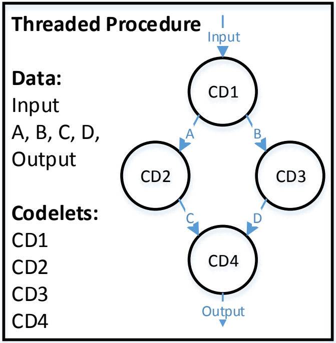
Threaded Procedures (TPs)

TPs are containers for codelet graphs, with additional meta-data.

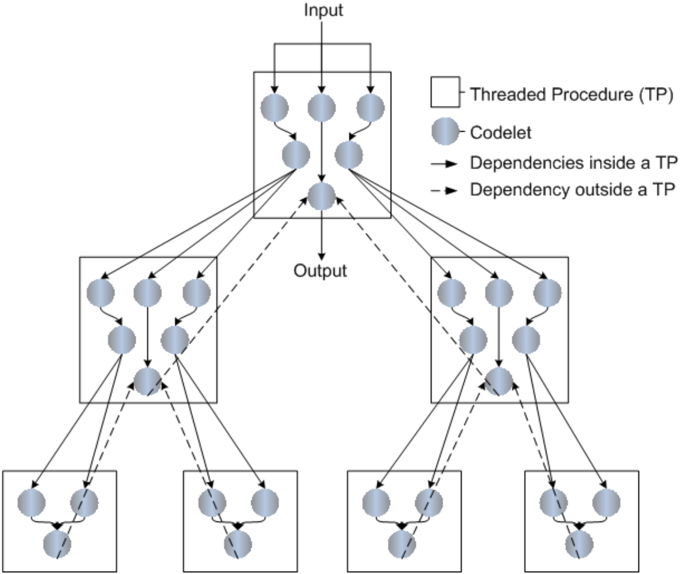
Description

- Invoked in a control-flow manner
- Called by a codelet from another CDG
- Feature a frame which contains the context of the CDG

See Stéphane Zuckerman, Suetterlein, et al. 2011



An Example of Computation Using Threaded Procedures



See Stéphane Zuckerman, Suetterlein, et al. 2011

Outline

2 DARTS: An Implementation of the Codelet Model

- DARTS: Implementation of the Codelet Machine Model
- DARTS: Experimental Results

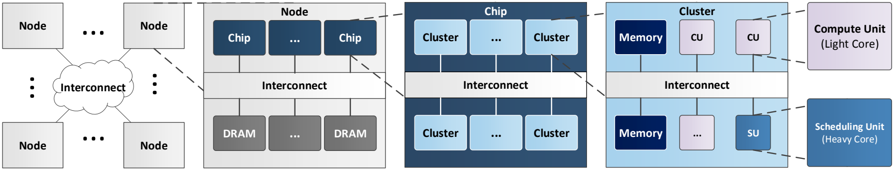
Objectives

- Faithfulness to the codelet execution model
- Modularity
  - So that portions of the runtime can be added or changed easily
  - For example: we have several codelet schedulers from which to choose
- Portability: Object-oriented, written in C++98, and makes use of open-source libraries:
  - `hwloc`: to determine the topology of the underlying system (HW threads/cores, caches, *etc.*)
  - If present on the system, it uses Intel TBB's lock-free queues

See Suetterlein, Stéphane Zuckerman, and GuangR. Gao 2013

DARTS: Implementation of the Codelet Machine Model

- Computation Units (CUs) embed a single producer/consumer ring buffer to store ready codelets
- Synchronization Units (SUs) embed two pools: Threaded Procedures and ready codelets.
- Heavy reliance on lock-free data structures
- SUs can temporarily assume the role of CUs if all other CUs are busy and there are ready codelets left to execute.



See Suetterlein, Stéphane Zuckerman, and GuangR. Gao 2013

Experimental Setup

AMD Opteron 6234 (Bulldozer) – Mills – 128 GiB DDR DRAM

		Cache Level	Shared By	Size (KiB)
Clock (GHz)	2.4	L1 Data	1 core	16
Threads / core	1	L1 Instruction	1 core	64
Cores / socket	12	L2 Unified	2 cores	2048
Sockets / node	4	L3 Unified	6 cores	6144
Compiler		gcc v4.6		
Math Library		AMD Core Math Library (ACML) v5.3		

Note: FPUs are shared between 2 cores.

Intel Xeon E5-2670 (Sandy Bridge) – FatNode – 64 GiB DDR3 DRAM

		Cache Level	Shared By	Size (KiB)
Clock (GHz)	2.6	L1 Data	2 threads	32
Threads / core	2	L1 Instruction	2 threads	32
Cores / socket	8	L2 Unified	2 threads	256
Sockets / node	2	L3 Unified	8 threads	20480
Compiler		gcc v4.7		
Math Library		Intel Math Kernel Library (MKL) v11.1		

Note: Functional units are shared between 2 threads.

Description of DGEMM

- **D**ouble precision **GE**neral **M**atrix **M**ultiplication
- Used ACML or MKL as sequential building blocks (no tiling/blocking, *etc.*, needed)
- We compared several codelet scheduling policies within a cluster of cores

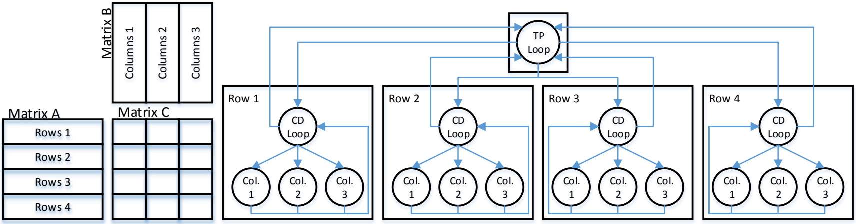


Figure: Our Codelet Graph decomposition for a parallel DGEMM

See Suetterlein, Stéphane Zuckerman, and GuangR. Gao 2013

Running DGEMM in DARTS – Mills – Strong Scaling

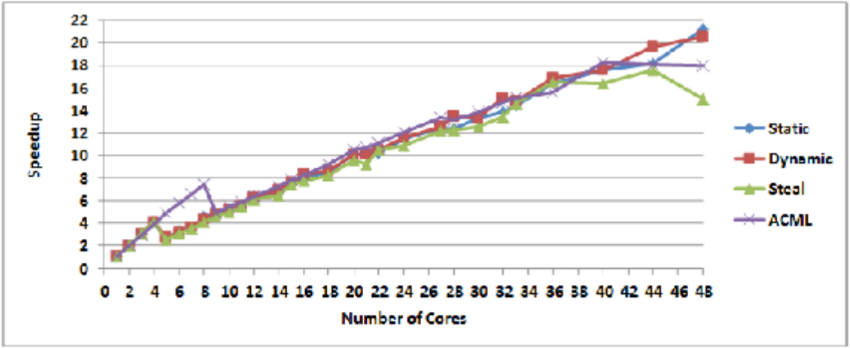


Figure: 10000×10000 Square DGEMM – Strong Scaling.

See Suetterlein, Stéphane Zuckerman, and GuangR. Gao 2013

Running DGEMM in DARTS – Mills – Weak Scaling

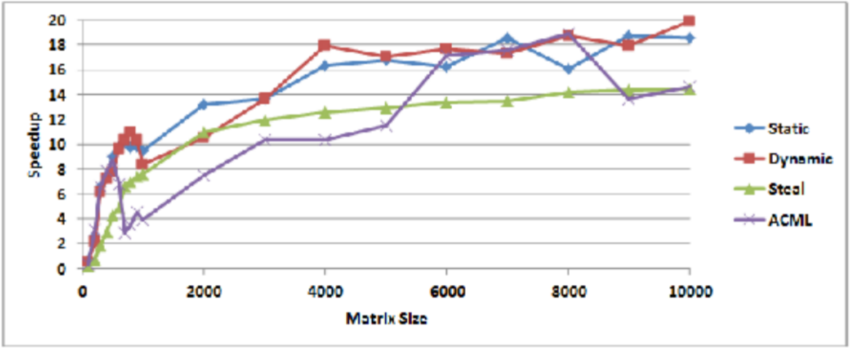


Figure: 48 cores – Square DGEMM – Weak Scaling.

See Suetterlein, Stéphane Zuckerman, and GuangR. Gao 2013

Running DGEMM in DARTS – FatNode – Strong Scaling

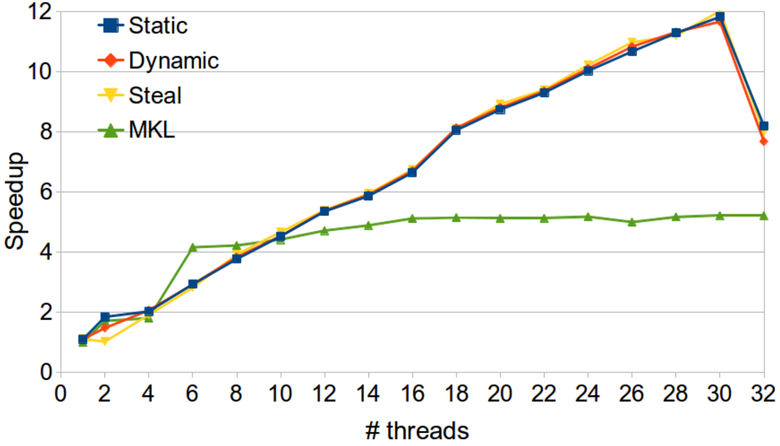


Figure: 3072 × 3072 Square DGEMM – Strong Scaling.

Running DGEMM in DARTS – FatNode – Weak Scaling

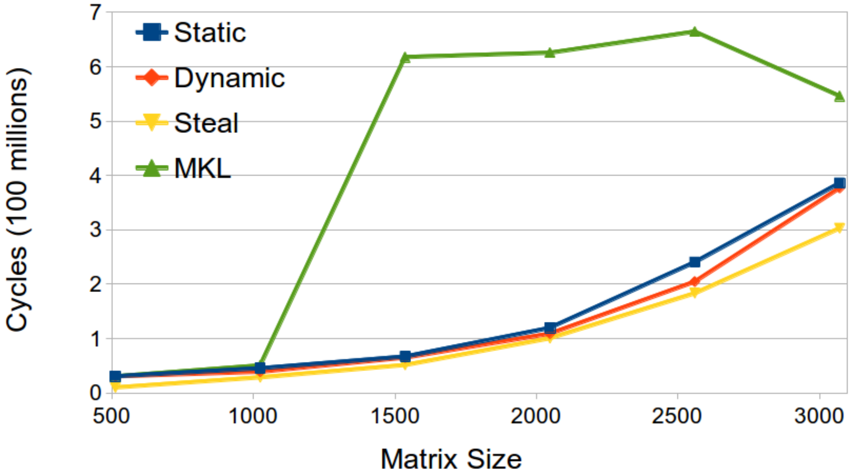
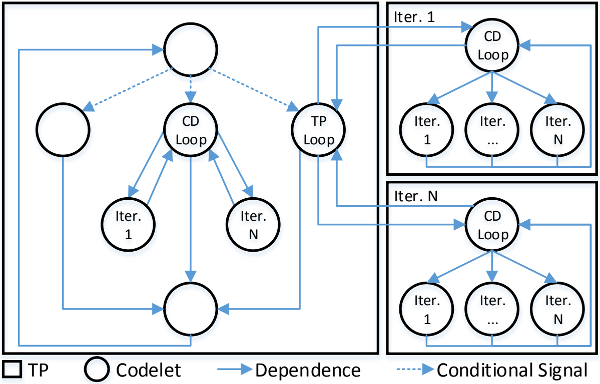


Figure: 32 threads – Square DGEMM – Weak Scaling.

Running Graph500 in DARTS – Codelet Graph

Description of Graph500

- Reused reference code (`http://graph500.org`)
- Only modified the breadth-first search phase (BFS)
- Compared with reference OpenMP parallelization
- Unit: Traversed Edges Per Second (TEPS)



See Suetterlein, Stéphane Zuckerman, and GuangR. Gao 2013

Running Graph500 in DARTS – Mills – Strong Scaling

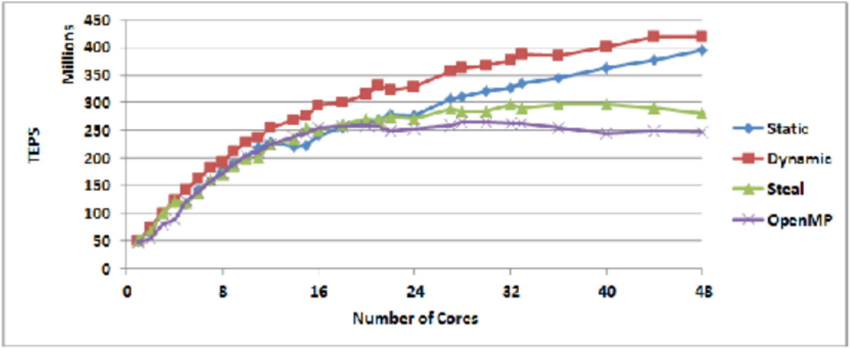


Figure: $\mathit{Scale} = 2^{18}$ – Graph500 – Strong Scaling

See Suetterlein, Stéphane Zuckerman, and GuangR. Gao 2013

Running Graph500 in DARTS – Mills – Weak Scaling

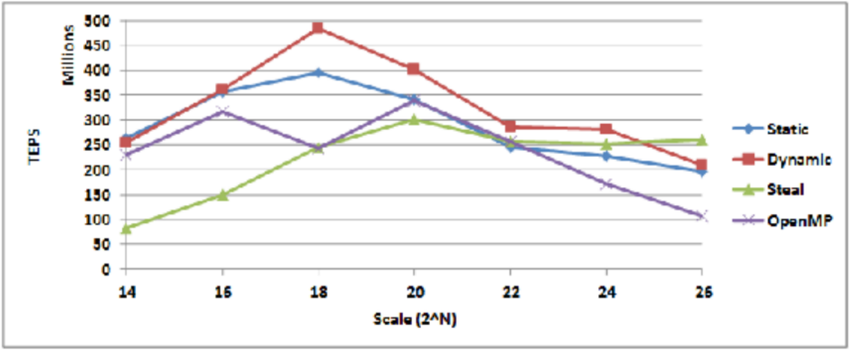


Figure: 48 cores – Graph500 – Weak Scaling

See Suetterlein, Stéphane Zuckerman, and GuangR. Gao 2013

Running Graph500 in DARTS – FatNode – Strong Scaling

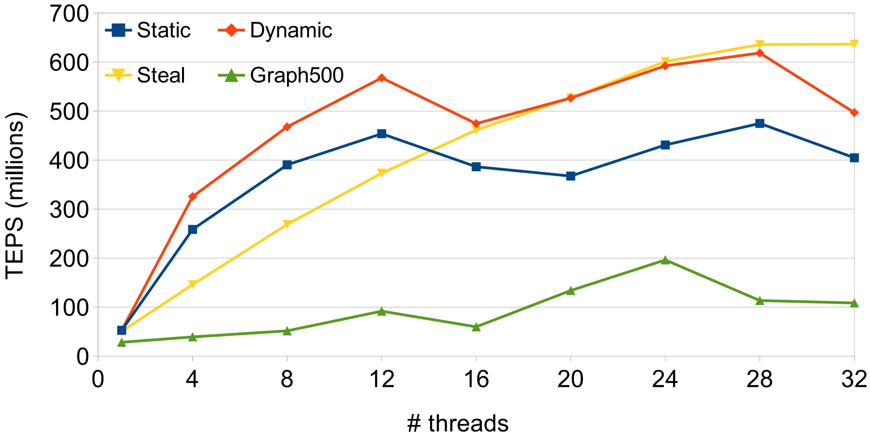


Figure: $Scale = 2^{18}$ – Graph500 – Strong Scaling

See Suetterlein, Stéphane Zuckerman, and GuangR. Gao 2013

Running Graph500 in DARTS – FatNode – Weak Scaling

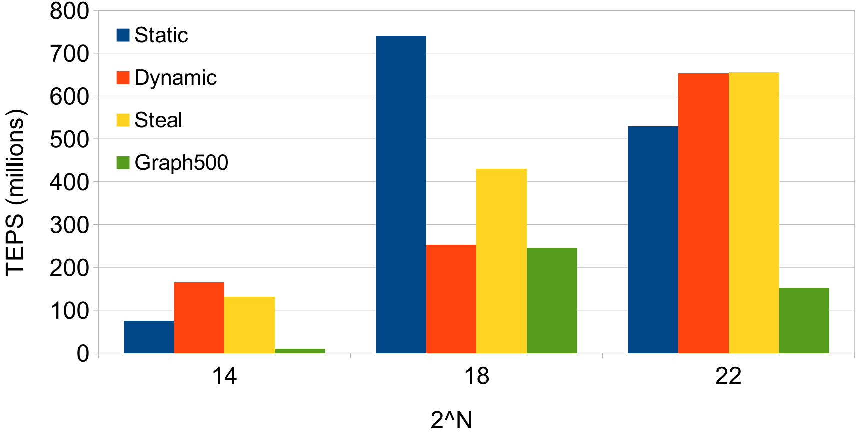


Figure: 32 cores – Graph500 – Weak Scaling

See Suetterlein, Stéphane Zuckerman, and GuangR. Gao 2013

Outline

3. **Examples of DARTS Programs**
   - Fibonacci Computation
   - DAXPY Computation

Fibonacci Numbers

Definition: Fibonacci Sequence

$$\begin{cases} U_0 = 0 \\ U_1 = 1 \\ U_n = U_{n-1} + U_{n-2} \end{cases}, \forall n \in \mathbb{N}$$

Definition: Fibonacci Sequence

$$\begin{cases} U_0 = 0 \\ U_1 = 1 \\ U_n = U_{n-1} + U_{n-2} \end{cases}, \forall n \in \mathbb{N}$$

```
int fib(int n) {
    return n < 2 ? n : fib(n-1) + fib(n-2);
}
```

Figure: Naïve Fibonacci Number Computation

```
int fib(int n) {
  int n1, n2;
  if (n < 2) /* cutoff value -- base case */
    return n;

  n1 = fib(n-1);
  n2 = fib(n-2);

  return n1+n2; /* Done -- we can return the sum */
}
```

Figure: Naïve Fibonacci Number Computation

Naïve Fibonacci Numbers Calculation in DARTS (1)

```
struct FibChecker : public Codelet {
    FibChecker(ThreadedProcedure* frame)
    : Codelet(0,0,frame,SHORTWAIT) {}
};
struct FibAdder : public Codelet {
    FibAdder(ThreadedProcedure* frame)
    : Codelet(2,2,frame,SHORTWAIT) {}
};
struct Fib : public ThreadedProcedure {
    int  n,                                              // input
    int  n1, n2,                                         // local to CDG
        *res;                                            // output
    FibChecker checker;                                  // checks if n < 2
    FibAdder   adder;                                    // returns n1+n2
    Codelet*   toSignal;                                 // signal when done
    Fib(int num, int* r,  Codelet* signalUp)             // ``Codelet 0''
    : n(num)                                             // input init
    , n1(0), n2(0)                                       // n1, n2 inits
    , res(r)                                             // output init
    {}
};
```

Naïve Fibonacci Numbers Calculation in DARTS (2)

```
void FibChecker::fire(void) {
    Fib      *frame     = static_cast<Fib*>(myTP_);
    int       n         = frame->n,
             *res       = frame->res;
    Codelet *toSignal = frame->toSignal;

    if (n < 2) { /* cutoff value -- base case */
        *res = n;
        toSignal->decDep();
        return;
    }
    frame->adder.add();
    invoke<Fib>(frame,n-1,&frame->n1,&frame->adder);
    invoke<Fib>(frame,n-2,&frame->n1,&frame->adder);
}
void FibAdder::fire(void) {
    Fib* frame = static_cast<Fib*>(myTP_);
    *(frame->res)      = frame->n1 + frame->n2;
    frame->toSignal->decDep();
    return;
}
```

Simplifying the Notation. . .

```
#define DEF_TP(TPName) struct TPName : public ThreadedProcedure
#define DEF_CODELET(name,dep,wait)                      \
 struct name : public darts::Codelet                    \
{                                                       \
    name(uint32_t deps=0, uint32_t reset=0,             \
         darts::ThreadedProcedure *frame,               \
         uint64_t meta=SHORTWAIT)                       \
    : darts::Codelet(deps,deps,frame,meta)              \
    {}                                                  \
    name(darts::ThreadedProcedure *frame)               \
    : darts::Codelet(dep,dep,frame,wait)                \
    {}                                                  \
    virtual void fire();                                \
}
#define LOAD_FRAME(TPType)  TPType* frame = (TPType*)myTP_
#define FRAME(field)        frame->field
#define INVOKE(TPType, ...) invoke<TPType>(frame,__VA_ARGS__)
#define ADD(field)          frame->field.add()
#define SIGNAL(field)       frame->field->decDep()
#define SYNC(field)         frame->field.decDep()
#define DARTS_EXIT()        darts::Runtime::finalSignal.decDep()
#define EXIT_TP()           return
```

Figure: DARTS Macros

Naïve Fibonacci Numbers Calculation in DARTS
More Compact

```
DEF_CODELET( FibChecker, 0, SHORTWAIT );
DEF_CODELET(   FibAdder, 2,  LONGWAIT );
DEF_TP(Fib) {
    int  n,                                          // input
    int  n1, n2,                                     // local to CDG
         *res;                                       // output
    FibChecker checker;                              // checks if n < 2
    FibAdder   adder;                                // returns n1+n2
    Codelet*   toSignal;                             // signal when done
    Fib(int num, int* r,  Codelet* signalUp)         // ``Codelet 0''
    : n(num)                                         // input init
    , n1(0), n2(0)                                   // n1, n2 inits
    , res(r)                                         // output init
    {}
};
```

Naïve Fibonacci Numbers Calculation in DARTS

More Compact

```
void FibChecker::fire(void) {
    LOAD_FRAME(Fib);
    int        n          = FRAME(n),
              *res        = FRAME(res);
    Codelet* toSignal = FRAME(toSignal);

    if (n < 2) { /* cutoff value -- base case */
        *res = n;
        toSignal->decDep();
        EXIT_TP();
    }
    ADD(adder);
    INVOKE(Fib,n-1,&FRAME(n1),&FRAME(adder));
    INVOKE(Fib,n-2,&FRAME(n2),&FRAME(adder));
}

void FibAdder::fire(void) {
    LOAD_FRAME(Fib);
    *( FRAME(res) ) = FRAME(n1) + FRAME(n2);
    SIGNAL(toSignal);
    EXIT_TP();
}
```

DAXPY

Definition: Double-precision Alpha-times-X-Plus-Y

$$\forall \alpha \in \mathbb{R}, \vec{X} \in \mathbb{R}^n, \vec{Y} \in \mathbb{R}^n : \vec{Y} = \alpha.\vec{X} + \vec{Y}$$

DAXPY

Definition: Double-precision Alpha-times-X-Plus-Y

$$\forall \alpha \in \mathbb{R}, \vec{X} \in \mathbb{R}^n, \vec{Y} \in \mathbb{R}^n : \vec{Y} = \alpha.\vec{X} + \vec{Y}$$

```
void DAXPY(const int     N, const double alpha,
           const double* X, const int    incX,
           double*       Y, const int    incY) {
    if (alpha == 0.0) // Y = Y + 0*X
        return;
    if (incX == 1 && incY == 1) {
        if (alpha == 1.0) // Y = Y + X
            daxpy_simple(Y,X,N);
        else
            daxpy_kernel(Y,X,alpha,N);
        return;
    }
    // General case
    int64_t ix = incX >= 0 ? 0 : -incX * (N-1),
            iy = incY >= 0 ? 0 : -incY * (N-1);
    for (int64_t i = 0; i < N; ++i, ix += incX, iy += incY)
        Y[iy] += alpha * X[ix];
}
```

Figure: Naïve DAXPY computation

DAXPY in DARTS

```
#define DEF_CODELET_ITER(name,dep,meta,id)                    \
class name : public Codelet {                                 \
private:                                                      \
    const uint64_t _id;                                       \
public:                                                       \
    name(uint32_t deps=0, uint32_t reset=0,                   \
         ThreadedProcedure* frame=0, uint64_t meta,           \
         const uint64_t id=0)                                 \
    : Codelet(deps,reset,frame,meta)                          \
    , _id(id)                                                 \
    {}                                                        \
    name(ThreadedProcedure* frame=0, const uint64_t id=0)     \
    : Codelet(dep,dep,frame,meta)                             \
    , _id(id)                                                 \
    {}                                                        \
    virtual void fire();                                      \
}
```

DAXPY in DARTS

Main DAXPY Function

```
DEF_CODELET(DaxpySpawn,0,SHORTWAIT);
DEF_CODELET(DaxpySync,2,LONGWAIT);
DEF_TP(Daxpy) {
    double*         Y;
    const double*   X;
    const double    alpha;
    const uint64_t  N;
    DaxpySpawn spawn;
    DaxpySync  sync;
    Codelet*   signalUp;

    Daxpy
    (
        double*       y, const double*  x,
        const double  a, const uint64_t n,
        Codelet* up
    )
    : Y(y), X(x), alpha(a), N(n)
    , spawn(this), sync(this), signalUp(up)
    {
        add(&spawn);
    }
};
```

DAXPY in DARTS I

Main DAXPY Function

```
#include "Daxpy.h"
const uint64_t DAXPY_CUTOFF = 0x100000; // 8M elements / CU
void DaxpySpawn::fire(void)  {
  LOAD_FRAME(Daxpy);
  double*        Y     = FRAME(Y);
  const double*  X     = FRAME(X);
  const double   alpha = FRAME(alpha);
  const uint64_t N     = FRAME(N);
  if  ( N < DAXPY_CUTOFF )     { // Sequential DAXPY:
    cblas_daxpy(N,alpha,X,1,Y,1);
    SIGNAL(signalUp);
  } else                        {
    ADD(sync);
    if  ( N < DAXPY_CUTOFF*(nCU+1) )                                  {
        INVOKE(DaxpyCDLoop, Y, X, alpha, N, &FRAME(sync));
        SYNC(sync);          // sync is waiting for 2 signals
    } else                                                            {
        INVOKE(Daxpy, Y,     X,     alpha,   N/2, &FRAME(sync));
        INVOKE(Daxpy, Y+N/2, X+N/2, alpha, N-N/2, &FRAME(sync));
    }
  }
  EXIT_TP();
}
```

DAXPY in DARTS II
Main DAXPY Function

```
void
DaxpySync::fire(void)
{
    LOAD_FRAME(Daxpy);
    SIGNAL(signalUp);
    EXIT_TP();
}
```

DAXPY in DARTS

DAXPY Codelet Loop Declaration

```
extern uint64_t nCU, nSU;
DEF_CODELET_ITER(DaxpyCDLoopCompute,0,SHORTWAIT);
DEF_CODELET(DaxpyCDLoopSync,2,LONGWAIT);
DEF_TP(DaxpyCDLoop) {
  double* Y; const double* X; const double alpha; const uint64_t N;
  DaxpyCDLoopCompute* compute;
  DaxpyCDLoopSync     sync;
  Codelet*            signalUp;
  DaxpyCDLoop(double*        y, const double*  x,
              const double   a, const uint64_t n, Codelet* up)
  : Y(y), X(x), alpha(a), N(n)
  , compute(new DaxpyCDLoopCompute [nCU+1])
  , sync(nCU+1,nCU+1,this,LONGWAIT), signalUp(up)
  {
    for ( size_t i = 0; i < nCU+1; ++i ) {
      compute[i] = DaxpyCDLoopCompute {0,0,this,SHORTWAIT,i};
      add( compute + i );
    }
  }
  ~DaxpyCDLoop() { delete [] compute; }
};
```

DAXPY in DARTS I

DAXPY Codelet Loop Definition

```
//----- DaxpyCDLoop.cpp -----//
#include "DaxpyCDLoop.h"
#include <cblas.h>

void DaxpyCDLoopCompute::fire(void) {
    LOAD_FRAME(DaxpyCDLoop);

    const uint64_t N      = FRAME(N),
                   cs     = N  / (nCU+1), // chunk size
                   rem    = N  % (nCU+1), // leftover iterations
                   off    = cs * getID(), // base offset
                   fin_cs = cs + (getID() == nCU ? rem : 0);

    cblas_daxpy(fin_cs,FRAME(alpha),FRAME(X)+off,1,FRAME(Y)+off,1);
    SIGNAL(signalUp);
    EXIT_TP();
}

void DaxpyCDLoopSync::fire(void) {
    LOAD_FRAME(DaxpyCDLoop);
    SIGNAL(signalUp);
    EXIT_TP();
```

DAXPY in DARTS II

DAXPY Codelet Loop Definition

```
}
```

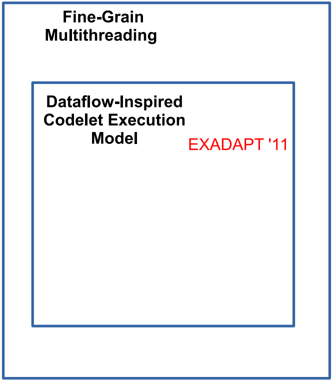
4 The Future of Codelets

- Self-Awareness and Codelets
- The SPARTA Program Execution Model

The Future of Codelets

The Story So Far

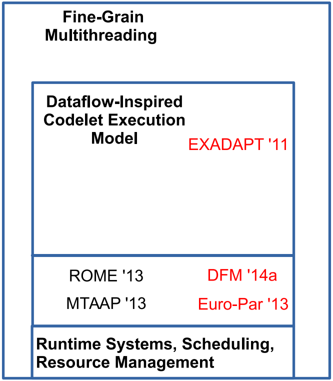
- We proposed the codelet execution model to answer the need for scalability, performance, energy efficiency, fault-tolerance, and programmability
- Experimental results show that the Codelet Model can be competitive with current multicore environments
- With hardware support, the Codelet Model displays very high potential to scale to large numbers of cores



The Future of Codelets

The Story So Far

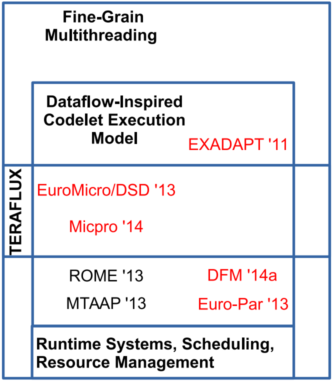
- We proposed the codelet execution model to answer the need for scalability, performance, energy efficiency, fault-tolerance, and programmability
- Experimental results show that the Codelet Model can be competitive with current multicore environments
- With hardware support, the Codelet Model displays very high potential to scale to large numbers of cores



The Future of Codelets

The Story So Far

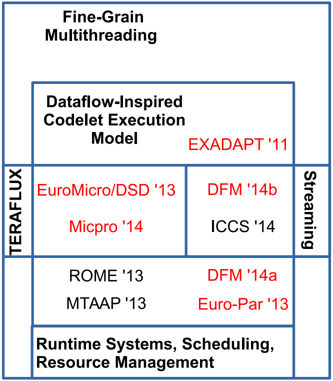
- We proposed the codelet execution model to answer the need for scalability, performance, energy efficiency, fault-tolerance, and programmability
- Experimental results show that the Codelet Model can be competitive with current multicore environments
- With hardware support, the Codelet Model displays very high potential to scale to large numbers of cores



The Future of Codelets

The Story So Far

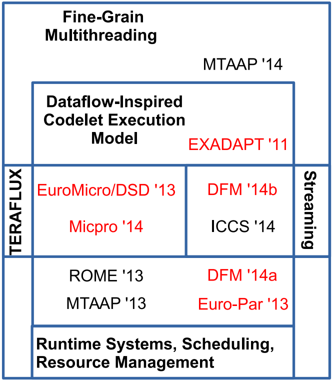
- We proposed the codelet execution model to answer the need for scalability, performance, energy efficiency, fault-tolerance, and programmability
- Experimental results show that the Codelet Model can be competitive with current multicore environments
- With hardware support, the Codelet Model displays very high potential to scale to large numbers of cores



The Future of Codelets

The Story So Far

- We proposed the codelet execution model to answer the need for scalability, performance, energy efficiency, fault-tolerance, and programmability
- Experimental results show that the Codelet Model can be competitive with current multicore environments
- With hardware support, the Codelet Model displays very high potential to scale to large numbers of cores



Outline

4. **The Future of Codelets**
   - Self-Awareness and Codelets
     - Introduction
     - Background
     - Extending the Codelet Specification
     - Combining Resource Usage Description with Hierarchical & Distributed Adaptation
     - Self-Aware Codelets: Conclusion
   - The SPARTA Program Execution Model

Introduction I

Current HPC Systems

- Combination of MPI and OpenMP for inter- and intra-node parallel computations.
- Inter-node communication is getting better by (small) increments
- The real revolution is happening at the compute node level
  - Presence of accelerators and co-processors: programming model shift already here

Projections for Future HPC Systems

- Performance is not the only important metric anymore–even at the compute node level
- Power and energy consumption are becoming more significant every day
- So is resiliency/fault-tolerance

Introduction II

This is where dataflow-inspired models can help, and in particular the Codelet Model.

Our Goal: To bridge the Codelet Model with a self-aware fine-grain resource management system

- Extend the codelet specification
- Combine codelet resource usage description with hierarchical and distributed adaptation to satisfy user goals

See Giorgi et al. 2014; Stéphane Zuckerman, Suetterlein, et al. 2011; Carter et al. 2013

Background

The Codelet Model: a Hybrid Event-Driven, von Neumann-Dataflow PXM

- Designed with extreme-scale systems in mind.
- Has been the target of several runtime systems implementations
- Inspiration behind the Intel/Rice driven Open Community Runtime (OCR)

See DARPA-BAA-10-37 2010-2012; Lauderdale and Khan 2012; Suetterlein, Stéphane Zuckerman, and GuangR. Gao 2013; Department of Energy 2012–2014; Hoffmann 2013; Landwehr, Stéphane Zuckerman, and G. R. Gao 2013.

Background

The Codelet Model: a Hybrid Event-Driven, von Neumann-Dataflow PXM

- Designed with extreme-scale systems in mind.
- Has been the target of several runtime systems implementations
- Inspiration behind the Intel/Rice driven Open Community Runtime (OCR)

Self-Awareness: Definition

A computer system is self-aware if it has (fine-grain) introspection capabilities, and has means to adapt and regulate its state (its "health") according to pre-defined goals—in particular, w.r.t. performance, energy, and resiliency.

See DARPA-BAA-10-37 2010-2012; Lauderdale and Khan 2012; Suetterlein, Stéphane Zuckerman, and GuangR. Gao 2013; Department of Energy 2012–2014; Hoffmann 2013; Landwehr, Stéphane Zuckerman, and G. R. Gao 2013.

Background

The Codelet Model: a Hybrid Event-Driven, von Neumann-Dataflow PXM

- Designed with extreme-scale systems in mind.
- Has been the target of several runtime systems implementations
- Inspiration behind the Intel/Rice driven Open Community Runtime (OCR)

Self-Awareness: Definition

A computer system is self-aware if it has (fine-grain) introspection capabilities, and has means to adapt and regulate its state (its "health") according to pre-defined goals—in particular, w.r.t. performance, energy, and resiliency.

Self-Awareness for Extreme-Scale: Use of Observe-Decide-Act Loops

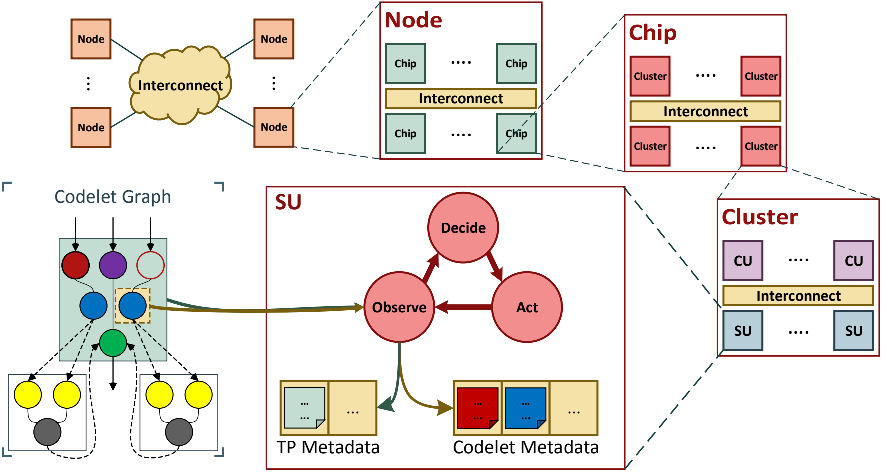
- Hoffmann showed how to apply ODA loops to current multicore systems
- We proposed using ODA loops for a Target Exascale Architecture
- The ODA loop must be hierarchical and distributed on extreme-scale systems

See DARPA-BAA-10-37 2010-2012; Lauderdale and Khan 2012; Suetterlein, Stéphane Zuckerman, and GuangR. Gao 2013; Department of Energy 2012–2014; Hoffmann 2013; Landwehr, Stéphane Zuckerman, and G. R. Gao 2013.

Adding Meta-Data to. . .

- Run programs with specific user goals
  - Example: "Target a 400W power consumption per chip, but be as parallel as possible."
- Drive the execution of threaded procedures
  - TPs are "codelet graph (CDG) containers"
  - Useful information can be stored w.r.t. the "shape" of the CDG
- Couple fine-grain resource management and codelets
  - Codelets themselves can carry useful resource usage information
  - Examples: Preferred computation device, no FP ops, bandwidth hungry, . . .
- Define mechanics of the threaded procedure
  - TP meta-data should be seen as "guiding rules"
  - Codelets meta-data supersede TP rules when they conflict

High-Level View of our Proposed System



Combining Resource Usage Description with Hierarchical & Distributed Adaptation

Going Further with Self-Awareness

- Meta-data defined by the compiler and/or hero programmers are not enough
- Each portion of the hardware needs to maintain a “meta-data cache” for recently executed TP/set of codelets.
- The end goal is to rapidly converge toward a stable state according to various (possibly conflicting) goals

Conclusion & Future Work I

Conclusion

- We propose a way to bridge fine-grain event-driven dataflow-inspired execution models with self-awareness aspects, using the Codelet Model
- Self-awareness is achieved through ODA loops
- Meta-data must be added both for coarse and fine grain levels of parallelism to help the system converge faster toward user and system goals
- The system must also maintain its own meta-data cache to deal with local resource management issues

Conclusion & Future Work II

Current Progress: Self-Aware Framework

- Models a single chip of our TEA machine, including an energy and a heat/temperature models
- A protocol to implement ODA loops for a hierarchical system
- It targets the OCR API

Future work includes

- Finish implementing several control policies for ODA, and evaluate them on varied (synthetic) workloads
- Add "horizontal" aspect (*i.e.*, not just hierarchical aggregation of "health")

The Evolution of Computer Systems

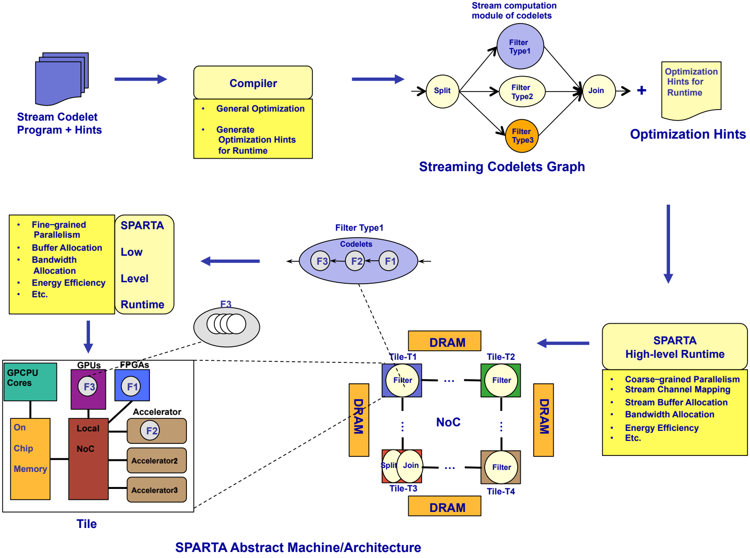
- Evolution from homogeneous single-core to heterogeneous multi-core systems
- This is largely due to vendors reaching new physical limits (power wall, heat wall, memory wall, *etc.*)
- Consequence : new programming and execution models must be designed to better exploit this wealth of available parallelism.

Introduction II

Three main problems to solve

- Multi-grain parallelism exploitation (fine and coarse)
- Take advantage of heterogeneous HW and workloads and types
- Develop efficient resource management mechanisms: favor locality and minimize data movement

SPARTA System Overview



Reusing a Fine-Grain Event-Driven Dataflow-Inspired Execution Model

- Use the Codelet Model fine-grain dataflow-inspired PXM as a basis.
- Extend codelets into Stream Codelets to account for buffer sizes and known latencies
- "Determinacy-by-default" but allows explicitly non-determinate regions of code

Extending the Codelet Model to Streams II

Helping with bulk data transfer latencies and scheduling

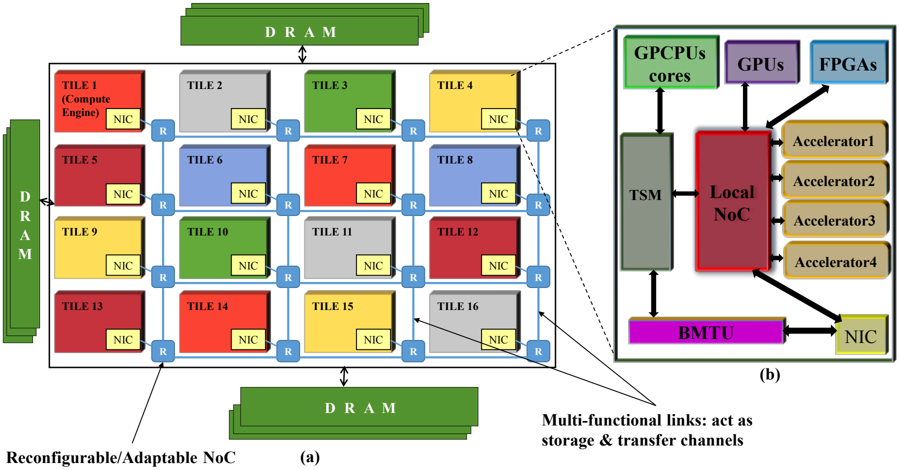
- Programs are partitioned into Stream Modules connected by inter-module channels
- SM: autonomous group of codelets connected by intra-module stream channels
  - Specific data consumption/production rates.
  - Ready if enough available data items and buffer space

The Streaming Codelet Model is Event-Driven

- Both streaming modules and codelets are event-driven
  - Data availability is the primary event (multi-grain parallelism)
  - SM level: pipelining and task parallelism
  - SC level: if connecting different portions of the HW in a tile, it needs stream channels; otherwise, simple signaling is enough

See Stéphane Zuckerman, Suetterlein, et al. 2011; Suetterlein, Stéphane Zuckerman, and GuangR. Gao 2013

SPARTA System Architecture Overview



Achieving End-to-End Energy Efficiency

- Memory sub-systems introduce energy overheads orders of magnitude higher than compute engines
- Both on- and off-chip memory systems memory systems must be optimized

A Tile-Based Heterogeneous & Hierarchical Architecture

- Features wide-range of customizable computing elements
  - From coarse-grain cores to fine-grain accelerators and FPGAs
  - Scalable reconfigurable high-performance interconnects
- Tiles are connected by global NoC; compute units by a local NoC

Fetching Entire Codelets and its Associated data

- Need bulk memory access (vs. word-at-a-time) $\rightarrow$ requires a change in the ISA
- Need a memory subsystem adapted to our goals: scratchpads and bulk memory transfer units within tiles

Other Areas of Exploration

- Evaluate different memory models (*e.g.*, I-Structures for heterogeneous computing)
- **Split-phase operations:** consumer-producer parallelism through asynchronous requests

Data Sharing: Low-Latency, High-Bandwidth, and Reconfigurable Interconnects

- **On/Off chip communications:** Major bottleneck for performance, energy, reliability
- **Compiler analysis:** Can partly address communication optimization
- **Hardware sensing:** Monitoring of traffic loads and communication behavior, network resource usage, power consumption, ...
- **Reconfigurable NoCs:** Use of dynamic reconfiguration algorithms to deal with fault-tolerance, energy efficiency, and provide capability for fast/energy efficient channels

See Lin et al. 2000; Weiss 2008; Shao, Jones, and Melhem 2006; DiTomaso, A. Kodi, and Louri 2014; A. K. Kodi, Louri, and Wang 2009

The SPARTA SysSW Management Tasks

- **Multigrain Parallelism:** Stream module and streaming codelet scheduling
- **Memory:** Stream module and channel creation (both between and within a tile)
- **Power/Energy usage:** Dynamic resource management through direct access to HW
- **Network traffic:** Detect contention between and within tiles

University of Delaware

Establishing an Efficient Communication Scheme

- **Hierarchical NoC Bandwidth Usage Management:** Channel assignment and bandwidth allocation
- **Compiler-Directed Asynchrony:** Determination of preferred computation engines for better load balancing and code placement

Finding the Right Hardware-Software Balance: Fast and Efficient Queues

- Part of the HW/SW trade-off is figuring out if queues should be implemented in HW or if the SW can be sufficient
- Potential intermediate approach: add atomic instructions better suited to queue handling (*i.e.*, not just `compare-and-swap`)

Related Work

Stream-Based Program Execution Models

- Have been a widely-studied field for the past 30 years
- Most relevant: Dennis' work on dataflow and Lee's SDF
- But: don't address heterogeneity and hierarchical systems, or perform only static partitioning

References

Nickolls et al. 2008; Che et al. 2008; Dennis 1974; E.A. Lee and Messerschmitt 1987; Govindarajan, G. Gao, and Desai 1994

Heterogeneous Multicore Architectures

- Heralded as solution to dark silicon and utilization wall issues
- Provide most compelling architectural path: Still provides continued performance scaling
- Architectures were very well explored but research tends to isolate a single dimension of the problem: Cores, Memory, or Interconnects

References

Suleman et al. 2011
Najaf-abadi and Rotenberg 2009

System Software Stack: Compiler & Runtime Techniques

- Optimize streams on specific architectures
- Map streaming languages to multithreaded processors
- None of these target highly heterogeneous systems, nor power and energy efficiency

References

Kapasi et al. 2001
Mattson et al. 2000
Kudlur and Mahlke 2008

To achieve high-performance on future extreme-scale computing systems, a complete overhaul must be performed.

Extending the Codelet Model to Streams

- On top of original multi-granularity and hierarchy, add:
  - Stream-related features (channels, buffer sizes)
  - Meta-data to help pick the best compute engine available

Novel Heterogeneous Tiled Architectures Exploration

- Must target end-to-end energy efficiency
- Tiles contain heterogeneous compute engines
- Hierarchical NoC: between and within tiles
- Memory must also be heterogeneous
- Needs reconfigurable interconnect fabric

Bridging Hardware & Software with Efficient System Software

- Must handle multigrain parallelism, memory, power/energy usage, and network traffic

5 Codelets: Conclusion

Codelets: a Summary

- The Codelet Model is a hybrid von Neumann/Dataflow model
  - Only fire when all their dependencies are satisfied
  - Non-preemptive
  - It inherits from the EARTH execution model
- They are not just data-driven: they are event-driven
  - Fine-grain resource management is necessary
- The codelet model was designed with extreme-scale in mind:
  - Need to extend the base model to take into account power/energy
  - Resiliency will eventually be an issue

Extending the Codelet Model (Future Work)

- Stream codelets
  - Extend Codelet Model to accelerators (ongoing work)
  - Restraining the Codelet Model to streams allows to make use of scratchpads and reduce problems due to the memory model
- Self-awareness
  - Dynamic codelet profiling to make power and energy aware scheduling decisions
  - Tagging of cores that fail too often under specific circumstances
- The Codelet Memory Model
  - Need to specify a memory model suitable for codelet execution

The End

The End

. . . For Now!

Arteaga, Jaime et al. (2014). “Position Paper: Locality-Driven Scheduling of Tasks for Data-Dependent Multithreading”. In: *Workshop on Multi-Threaded Architectures and Applications (MTAAP 2014)*. Phoenix, USA.

Arvind and David E. Culler (1986). “Dataflow Architectures”. In: *Annual Review of Computer Science* 1.1, pp. 225–253. DOI: 10.1146/annurev.cs.01.060186.001301. URL: http://dx.doi.org/10.1146/annurev.cs.01.060186.001301.

Arvind and Kim P. Gostelow (1982). “The U-Interpreter”. In: *IEEE Computer* 15.2, pp. 42–49. DOI: 10.1109/MC.1982.1653940. URL: http://doi.ieeecomputersociety.org/10.1109/MC.1982.1653940.

Carter, Nicholas P et al. (2013). "Runnemede: An Architecture for Ubiquitous High-Performance Computing". In: HPCA. Shenzhen, China.

Che, Shuai et al. (2008). "A performance study of general-purpose applications on graphics processors using CUDA". In: *Journal of Parallel and Distributed Computing* 68.10, pp. 1370–1380. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2008.05.014. URL: http://www.sciencedirect.com/science/article/pii/S0743731508000932 (visited on 02/15/2013).

Chen, Chen et al. (2013). "Towards Memory-Load Balanced Fast Fourier Transformations in Fine-Grain Execution Models." In: *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, Cambridge, MA, USA, May 20-24, 2013*. IEEE, pp. 1607–1617. DOI: 10.1109/IPDPSW.2013.47. URL: http://dx.doi.org/10.1109/IPDPSW.2013.47.

DARPA-BAA-10-37 (2010-2012). "UHPC: Ubiquitous High Performance Computing". In: Arlington VA, USA: DARPA.

Dennis, Jack B. (1974). "First version of a data flow procedure language". In: *Programming Symposium, Proceedings Colloque sur la Programmation, Paris, France, April 9-11, 1974*. Ed. by Bernard Robinet. Vol. 19. Lecture Notes in Computer Science. Springer, pp. 362–376. ISBN: 3-540-06859-7. DOI: 10.1007/3-540-06859-7_145. URL: http://dx.doi.org/10.1007/3-540-06859-7_145.

Dennis, Jack B., John B. Fosseen, and John P. Linderman (1972). "Data flow schemas". In: *International Sympoisum on Theoretical Programming, Novosibirsk, Russia, August 7-11, 1972, Proceedings*. Ed. by Andrei P. Ershov and V. A. Nepomniaschy. Vol. 5. Lecture Notes in Computer Science. Springer, pp. 187–216. ISBN: 3-540-06720-5. DOI: 10.1007/3-540-06720-5_15. URL: http://dx.doi.org/10.1007/3-540-06720-5_15.

Dennis, Jack B. and David Misunas (1974). "A Preliminary Architecture for a Basic Data Flow Processor". In: *Proceedings of the 2nd Annual Symposium on Computer Architecture, December 1974*. Ed. by Willis K. King and Oscar N. Garcia. ACM, pp. 126–132. DOI: 10.1145/642089.642111. URL: http://doi.acm.org/10.1145/642089.642111.

Department of Energy (2012–2014). *X-Stack — Extreme Scale Software Stack*. URL: http://www.xstack.org.

DiTomaso, Dominic, Avinash Kodi, and Ahmed Louri (2014). "QORE: A Fault-Tolerant Network-on-Chip Architecture with Power-Efficient Quad Function Channel (QFC) Buffers". In: *Accepted to appear in 20th IEEE International Symposium on High-Performance Computer Architecture (HPCA), Orlando, FL, February 15-19, 2014*. Orlando, FL, USA.

UNIVERSITY OF DELAWARE

Gao, Guang R., Joshua Suetterlein, and Stéphane Zuckerman (2011). *Toward an Execution Model for Extreme-Scale Systems-Runnemede and Beyond*. Technical Memo 104. University of Delaware, 140 Evans Hall, Newark,DE 19716: Computer Architecture & Parallel Systems Laboratory, Electrical & Computer Engineering Departement.

Giorgi, Roberto et al. (2014). "TERAFLUX: Harnessing dataflow in next generation teradevices". In: *Microprocessors and Microsystems* 38.8, pp. 976–990. ISSN: 0141-9331. DOI: http://dx.doi.org/10.1016/j.micpro.2014.04.001. URL: http://www.sciencedirect.com/science/article/pii/S0141933114000490.

Govindarajan, R., G. Gao, and P. Desai (1994). "Minimizing Memory Re-quirements in Rate-optimal Schedules". In: *In ASAP '94: Proceedings of the 1994 International Conference on Application Specific Array Processors*, pp. 75–86.

Hoffmann, Henry (2013). "SEEC: A Framework for Self-Aware Management of Goals and Constraints in Computing Systems". PhD thesis. Massachusetts Institute of Technology.

Kapasi, Ujval J. et al. (2001). "Stream Scheduling". In: *in Proceedings of the 3rd Workshop on Media and Streaming Processors*, pp. 82–92.

Kodi, Avinash Karanth, Ahmed Louri, and Janet Meiling Wang (2009). "Design of Energy-Efficient Channel Buffers with Router Bypassing for Network-on-Chips (NoCs)". In: *ISQED*, pp. 826–832.

Kudlur, Manjunath and Scott Mahlke (2008). "Orchestrating the execution of stream programs on multicore platforms". In: *SIGPLAN Not.* 43.6, pp. 114–124. ISSN: 0362-1340. DOI: 10.1145/1379022.1375596. URL: http://doi.acm.org/10.1145/1379022.1375596.

UNIVERSITY OF DELAWARE

Landwehr, Aaron, Stéphane Zuckerman, and Guang R. Gao (2013). "Toward a Self-aware System for Exascale Architectures." In: *Euro-Par 2013: Parallel Processing Workshops - BigDataCloud, DIHC, FedICI, HeteroPar, HiBB, LSDVE, MHPC, OMHI, PADABS, PROPER, Resilience, ROME, and UCHPC 2013, Aachen, Germany, August 26-27, 2013. Revised Selected Papers*. Ed. by Dieter an Mey et al. Vol. 8374. Lecture Notes in Computer Science. Springer, pp. 812–822. ISBN: 978-3-642-54419-4. DOI: 10.1007/978-3-642-54420-0_79. URL: http://dx.doi.org/10.1007/978-3-642-54420-0_79.

Lauderdale, Christopher and Rishi Khan (2012). "Position paper: Toward a codelet-based runtime for exascale computing". In: *Proceedings of the 2nd International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era*. EXADAPT '12. San Jose, California: ACM. ISBN: 978-1-4503-0708-6.

Lee, E.A. and D.G. Messerschmitt (1987). "Synchronous data flow". In: *Proceedings of the IEEE* 75.9, pp. 1235–1245. ISSN: 0018-9219. DOI: 10.1109/PROC.1987.13876.

Lee, E.A and D.G. Messerschmitt (1987). "Synchronous data flow". In: *Proceedings of the IEEE* 75.9, pp. 1235–1245. ISSN: 0018-9219. DOI: 10.1109/PROC.1987.13876.

Lin, W-Y. et al. (2000). "Performance Analysis of the I-Structure Software Cache on Multi-Threading Systems". In: *Proceedings of the 19th IEEE International Performance, Computing and Communication Conference, IPCCC2000*. Phoenix, Arizona, pp. 83–89.

Mattson, Peter et al. (2000). "Communication scheduling". In: *SIGARCH Comput. Archit. News* 28.5, pp. 82–92. ISSN: 0163-5964. DOI: 10.1145/378995.379005. URL: http://doi.acm.org/10.1145/378995.379005.

Najaf-abadi, H.H. and E. Rotenberg (2009). "Architectural Contesting". In: *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pp. 189–200. DOI: 10.1109/HPCA.2009.4798254.

Nickolls, John et al. (2008). "Scalable Parallel Programming with CUDA". In: *Queue* 6.2, pp. 40–53. ISSN: 1542-7730. DOI: 10.1145/1365490.1365500. URL: http://doi.acm.org/10.1145/1365490.1365500 (visited on 02/15/2013).

Papadopoulos, Gregory M. and David E. Culler (1990). "Monsoon: An Explicit Token-store Architecture". In: *Proceedings of the 17th Annual International Symposium on Computer Architecture*. ISCA '90. Seattle, Washington, USA: ACM, pp. 82–91. ISBN: 0-89791-366-3. DOI: 10.1145/325164.325117. URL: http://doi.acm.org/10.1145/325164.325117.

Shao, S., A.K. Jones, and R. Melhem (2006). "A compiler-based communication analysis approach for multiprocessor systems". In: *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, p. 10. DOI: 10.1109/IPDPS.2006.1639322.

Solinas, Marco et al. (2013). "The TERAFLUX Project: Exploiting the DataFlow Paradigm in Next Generation Teradevices." In: *2013 Euromicro Conference on Digital System Design, DSD 2013, Los Alamitos, CA, USA, September 4-6, 2013*. IEEE, pp. 272–279. DOI: 10.1109/DSD.2013.39. URL: http://dx.doi.org/10.1109/DSD.2013.39.

Suetterlein, Joshua, Stéphane Zuckerman, and GuangR. Gao (2013). “An Implementation of the Codelet Model”. In: *Euro-Par 2013 Parallel Processing*. Ed. by Felix Wolf, Bernd Mohr, and Dieter an Mey. Vol. 8097. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 633–644. ISBN: 978-3-642-40046-9. DOI: 10.1007/978-3-642-40047-6_63.

Suleman, M.A. et al. (2011). “Data Marshaling for Multicore Systems”. In: *Micro, IEEE* 31.1, pp. 56–64. ISSN: 0272-1732. DOI: 10.1109/MM.2010.105.

Watson, Ian and John R. Gurd (1982). “A Practical Data Flow Computer”. In: *Computer* 15.2, pp. 51–57. ISSN: 0018-9162. DOI: 10.1109/MC.1982.1653941.

Wei, Haitao et al. (2014). "A Dataflow Programming Language and its Compiler for Streaming Systems." In: *Proceedings of the International Conference on Computational Science, ICCS 2014, Cairns, Queensland, Australia, 10-12 June, 2014*. Ed. by David Abramson et al. Vol. 29. Procedia Computer Science. Elsevier, pp. 1289–1298. DOI: 10.1016/j.procs.2014.05.116. URL: http://dx.doi.org/10.1016/j.procs.2014.05.116.

Weiss, Ron (2008). "NSF Workshop on Emerging Models and Technologies in Computing: Bio-Inspired Computing and the Biology and Computer Science Interface." PhD thesis. United States Naval Academy.

Zuckerman, Stéphane, Jaime Arteaga, et al. (2014). *D9.3 – Evaluation of the Codelet Runtime System on a Teradevice*. Deliverable 9.3. University of Delaware, 140 Evans Hall, Newark,DE 19716: Computer Architecture & Parallel Systems Laboratory, Electrical & Computer Engineering Departement.

Zuckerman, Stéphane, Aaron Landwehr, et al. (2014). "Toward a Self-Aware Codelet Execution Model". In: *Proceedings of the Workshop on DataFlow Models for extreme-scale computing (DFM'14)*. Edmonton, AB, Canada.

Zuckerman, Stéphane, Joshua Suetterlein, et al. (2011). "Using a "Codelet" Program Execution Model for Exascale Machines: Position Paper". In: *Proceedings of the 1st International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era*. EXADAPT '11. San Jose, California: ACM, pp. 64–69. ISBN: 978-1-4503-0708-6. DOI: 10.1145/2000417.2000424. URL: http://doi.acm.org/10.1145/2000417.2000424.

Zuckerman, Stéphane, Haitao Wei, et al. (2014). "A Holistic Dataflow-Inspired System Design". In: *Proceedings of the Workshop on DataFlow Models for extreme-scale computing (DFM'14)*. Edmonton, AB, Canada.

References – Codelet Model I
Specification and Implementation

- Guang R. Gao, Joshua Suetterlein, and Stéphane Zuckerman (2011). *Toward an Execution Model for Extreme-Scale Systems-Runnemede and Beyond*. Technical Memo 104. University of Delaware, 140 Evans Hall, Newark,DE 19716: Computer Architecture & Parallel Systems Laboratory, Electrical & Computer Engineering Departement
- Stéphane Zuckerman, Joshua Suetterlein, et al. (2011). "Using a "Codelet" Program Execution Model for Exascale Machines: Position Paper". In: *Proceedings of the 1st International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era*. EXADAPT '11. San Jose, California: ACM, pp. 64–69. ISBN: 978-1-4503-0708-6. DOI: 10.1145/2000417.2000424. URL: http://doi.acm.org/10.1145/2000417.2000424
- Christopher Lauderdale and Rishi Khan (2012). "Position paper: Toward a codelet-based runtime for exascale computing". In: *Proceedings of the 2nd International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era*. EXADAPT '12. San Jose, California: ACM. ISBN: 978-1-4503-0708-6
- Joshua Suetterlein, Stéphane Zuckerman, and GuangR. Gao (2013). "An Implementation of the Codelet Model". In: *Euro-Par 2013 Parallel Processing*. Ed. by Felix Wolf, Bernd Mohr, and Dieter an Mey. Vol. 8097. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 633–644. ISBN: 978-3-642-40046-9. DOI: 10.1007/978-3-642-40047-6_63

References – Codelet Model II
Specification and Implementation

- Haitao Wei et al. (2014). "A Dataflow Programming Language and its Compiler for Streaming Systems." In: *Proceedings of the International Conference on Computational Science, ICCS 2014, Cairns, Queensland, Australia, 10-12 June, 2014*. Ed. by David Abramson et al. Vol. 29. Procedia Computer Science. Elsevier, pp. 1289–1298. DOI: 10.1016/j.procs.2014.05.116. URL: http://dx.doi.org/10.1016/j.procs.2014.05.116
- Stéphane Zuckerman, Aaron Landwehr, et al. (2014). "Toward a Self-Aware Codelet Execution Model". In: *Proceedings of the Workshop on DataFlow Models for extreme-scale computing (DFM'14)*. Edmonton, AB, Canada
- Stéphane Zuckerman, Haitao Wei, et al. (2014). "A Holistic Dataflow-Inspired System Design". In: *Proceedings of the Workshop on DataFlow Models for extreme-scale computing (DFM'14)*. Edmonton, AB, Canada

- Marco Solinas et al. (2013). "The TERAFLUX Project: Exploiting the DataFlow Paradigm in Next Generation Teradevices." In: *2013 Euromicro Conference on Digital System Design, DSD 2013, Los Alamitos, CA, USA, September 4-6, 2013*. IEEE, pp. 272–279. DOI: 10.1109/DSD.2013.39. URL: http://dx.doi.org/10.1109/DSD.2013.39
- Roberto Giorgi et al. (2014). "TERAFLUX: Harnessing dataflow in next generation teradevices". In: *Microprocessors and Microsystems* 38.8, pp. 976–990. ISSN: 0141-9331. DOI: http://dx.doi.org/10.1016/j.micpro.2014.04.001. URL: http://www.sciencedirect.com/science/article/pii/S0141933114000490
- Stéphane Zuckerman, Jaime Arteaga, et al. (2014). *D9.3 – Evaluation of the Codelet Runtime System on a Teradevice*. Deliverable 9.3. University of Delaware, 140 Evans Hall, Newark,DE 19716: Computer Architecture & Parallel Systems Laboratory, Electrical & Computer Engineering Departement

Other Dataflow and Data-Driven Related Work

- Jaime Arteaga et al. (2014). "Position Paper: Locality-Driven Scheduling of Tasks for Data-Dependent Multithreading". In: *Workshop on Multi-Threaded Architectures and Applications (MTAAP 2014)*. Phoenix, USA
- Chen Chen et al. (2013). "Towards Memory-Load Balanced Fast Fourier Transformations in Fine-Grain Execution Models." In: *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, Cambridge, MA, USA, May 20-24, 2013*. IEEE, pp. 1607–1617. DOI: 10.1109/IPDPSW.2013.47. URL: http://dx.doi.org/10.1109/IPDPSW.2013.47
- Aaron Landwehr, Stéphane Zuckerman, and Guang R. Gao (2013). "Toward a Self-aware System for Exascale Architectures." In: *Euro-Par 2013: Parallel Processing Workshops - BigDataCloud, DIHC, FedICI, HeteroPar, HiBB, LSDVE, MHPC, OMHI, PADABS, PROPER, Resilience, ROME, and UCHPC 2013, Aachen, Germany, August 26-27, 2013. Revised Selected Papers*. Ed. by Dieter an Mey et al. Vol. 8374. Lecture Notes in Computer Science. Springer, pp. 812–822. ISBN: 978-3-642-54419-4. DOI: 10.1007/978-3-642-54420-0_79. URL: http://dx.doi.org/10.1007/978-3-642-54420-0_79

Other References I

- Arvind and Kim P. Gostelow (1982). "The U-Interpreter". In: *IEEE Computer* 15.2, pp. 42–49. DOI: 10.1109/MC.1982.1653940. URL: http://doi.ieeecomputersociety.org/10.1109/MC.1982.1653940
- Jack B. Dennis, John B. Fosseen, and John P. Linderman (1972). "Data flow schemas". In: *International Sympoisum on Theoretical Programming, Novosibirsk, Russia, August 7-11, 1972, Proceedings*. Ed. by Andrei P. Ershov and V. A. Nepomniaschy. Vol. 5. Lecture Notes in Computer Science. Springer, pp. 187–216. ISBN: 3-540-06720-5. DOI: 10.1007/3-540-06720-5_15. URL: http://dx.doi.org/10.1007/3-540-06720-5_15
- Jack B. Dennis (1974). "First version of a data flow procedure language". In: *Programming Symposium, Proceedings Colloque sur la Programmation, Paris, France, April 9-11, 1974*. Ed. by Bernard Robinet. Vol. 19. Lecture Notes in Computer Science. Springer, pp. 362–376. ISBN: 3-540-06859-7. DOI: 10.1007/3-540-06859-7_145. URL: http://dx.doi.org/10.1007/3-540-06859-7_145
- Jack B. Dennis and David Misunas (1974). "A Preliminary Architecture for a Basic Data Flow Processor". In: *Proceedings of the 2nd Annual Symposium on Computer Architecture, December 1974*. Ed. by Willis K. King and Oscar N. Garcia. ACM, pp. 126–132. DOI: 10.1145/642089.642111. URL: http://doi.acm.org/10.1145/642089.642111
- Ian Watson and John R. Gurd (1982). "A Practical Data Flow Computer". In: *Computer* 15.2, pp. 51–57. ISSN: 0018-9162. DOI: 10.1109/MC.1982.1653941

Other References II

- Arvind and David E. Culler (1986). "Dataflow Architectures". In: *Annual Review of Computer Science* 1.1, pp. 225–253. DOI: 10.1146/annurev.cs.01.060186.001301. URL: http://dx.doi.org/10.1146/annurev.cs.01.060186.001301
- E.A Lee and D.G. Messerschmitt (1987). "Synchronous data flow". In: *Proceedings of the IEEE* 75.9, pp. 1235–1245. ISSN: 0018-9219. DOI: 10.1109/PROC.1987.13876
- Gregory M. Papadopoulos and David E. Culler (1990). "Monsoon: An Explicit Token-store Architecture". In: *Proceedings of the 17th Annual International Symposium on Computer Architecture*. ISCA '90. Seattle, Washington, USA: ACM, pp. 82–91. ISBN: 0-89791-366-3. DOI: 10.1145/325164.325117. URL: http://doi.acm.org/10.1145/325164.325117
- DARPA-BAA-10-37 (2010-2012). "UHPC: Ubiquitous High Performance Computing". In: Arlington VA, USA: DARPA
- Nicholas P Carter et al. (2013). "Runnemede: An Architecture for Ubiquitous High-Performance Computing". In: HPCA. Shenzhen, China
- **Hoffmann13**
- Department of Energy (2012–2014). *X-Stack — Extreme Scale Software Stack*. URL: http://www.xstack.org