CPEG 852 — Advanced Topics in Computing Systems
The EARTH Program Execution Model
Hybrid von Neumann/Dataflow Models

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

October 20-27, 2015

Outline

Outline

Outline

EARTH: The Secret Origins

- At the beginning of the 1990's, dataflow machines are considered obsolete compared to their vector and superscalar cousins
- Some people still think dataflow as a founding principle for models of computation is still sound
- Idea: mix RISC-like processors for the low-level execution and dataflow semantics for high-level concepts
  - Technically, this is more or less the definition of *macro-dataflow*
- EARTH is a bit different from macro-dataflow – see next slide

A Short Detour: Macro-Dataflow I

Macro-Dataflow: a Description

- Concept: take a sequence of instructions, and group them into a macro-dataflow actor
- A macro-dataflow actor has input arcs (for the "tokens") and output arcs
- A macro-dataflow actor has no state except the one formed by its input tokens
- Firing rule: when all input arcs have a token present, the macro-dataflow actor may fire
- A macro-dataflow actor always places tokens on its ouputs arcs *all at once*

Why EARTH Does Not Quite Follow a Macro-Dataflow Model

- In EARTH, actors *do* execute an uninterruptible sequence of instructions (like in macro-dataflow)
- But an EARTH actor can signal/produce data at any time during its execution
  - It is more "loose" in its asynchrony

What Is a Thread in EARTH?

- Parallel function invocation (threaded function invocation)
- A code sequence defined (by the user or a compiler) to be a thread (fiber)
- Usually, a threaded function's body will be partitioned into multiple threads/fibers

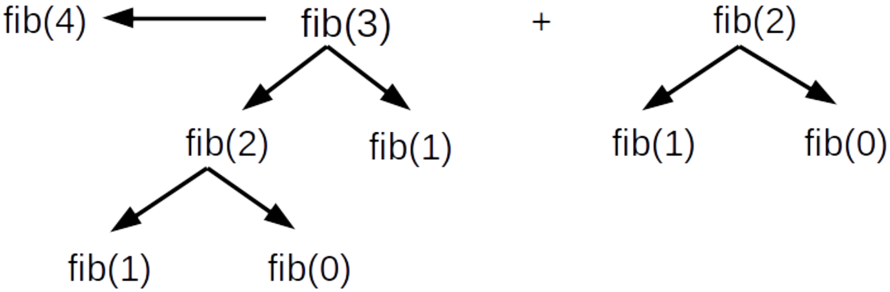
Outline

A Simple Example: Naïve Fibonacci

```c
u64  fib(u64 n) {
    if (n < 2) return n;
    return fib(n-1) + fib(n-2);
}
```

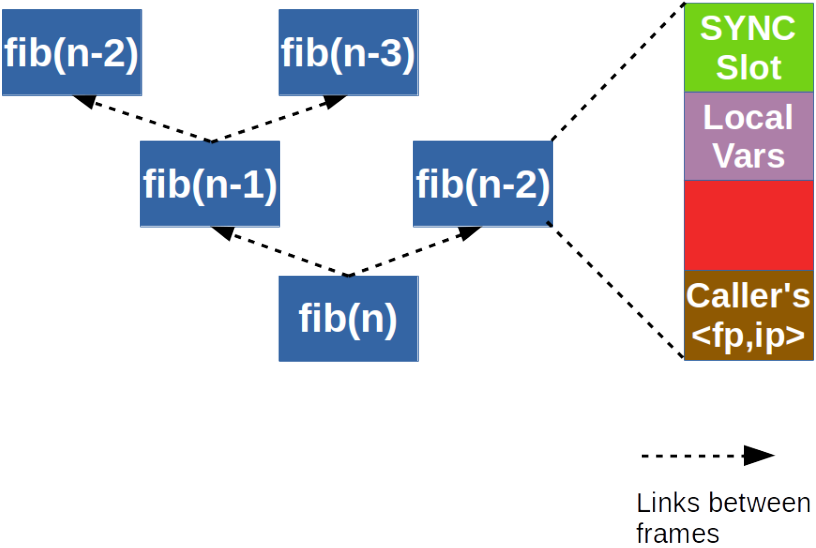
A Simple Example: Naïve Fibonacci

Computation Tree Example



A Simple Example: Naïve Fibonacci

Activation Frame Tree



Known Short Latencies, Known Long Latencies, and Unknown Latencies

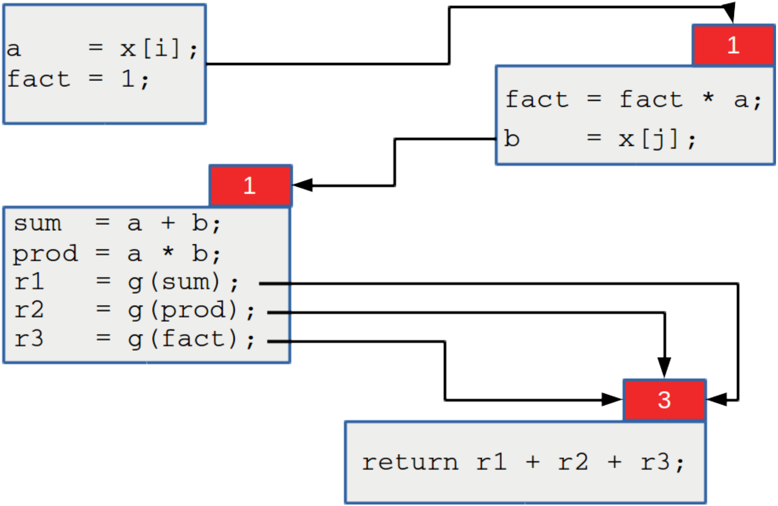
An Example

```
int  f(int *x, int i, int j) {
    int a, b, sum, prod, fact;
    int r1, r2, r3;
    a    = x[i];
    fact = 1;
    fact = fact * a;
```

```
    b    = x[j];
    sum  = a + b;
    prod = a * b;
    r1   = g(sum);
    r2   = g(prod);
    r3   = g(fact);

    return r1 + r2 + r3;
}
```

Known Short Latencies, Known Long Latencies, and Unknown Latencies

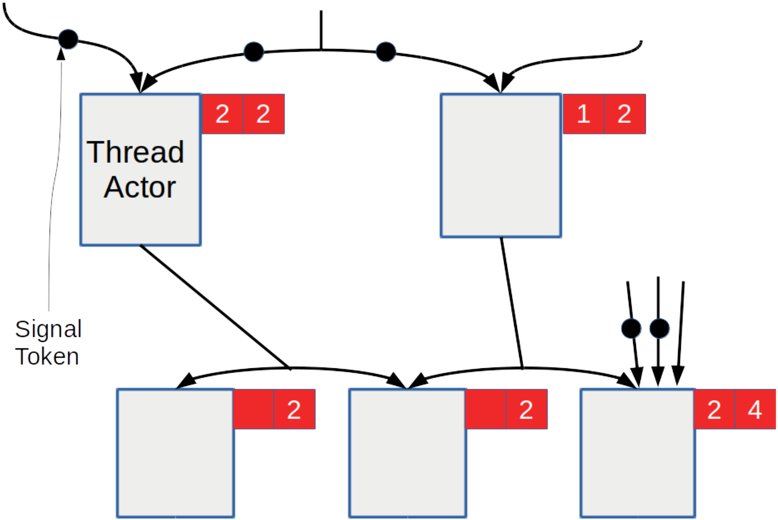
Partitioning Into Fibers



Fibers

- A fiber shares its enclosing "frame" with other fibers within the same threaded function invocation
- The state of a fiber includes:
  - Its instruction pointer
  - Its "temporary register set"
- A fiber is "ultra-lightweight:" it does not need dynamic storage (frame) allocation.
- Our focus: non-preemptive threads – which we call fibers

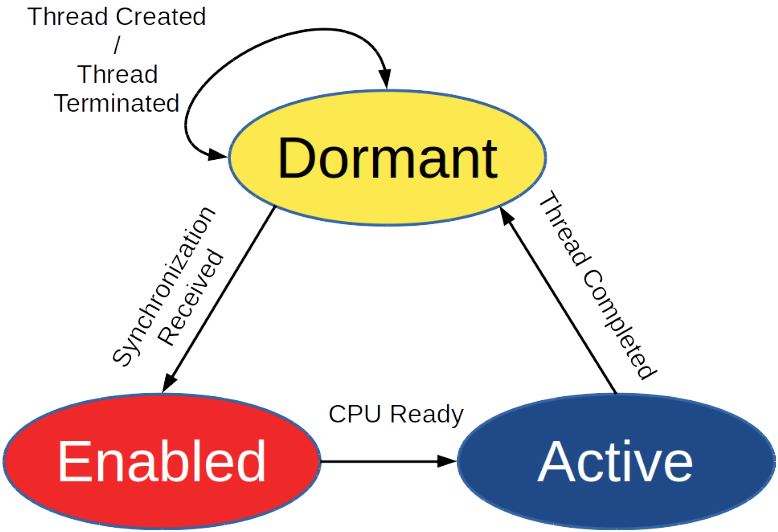
The EARTH Execution Model



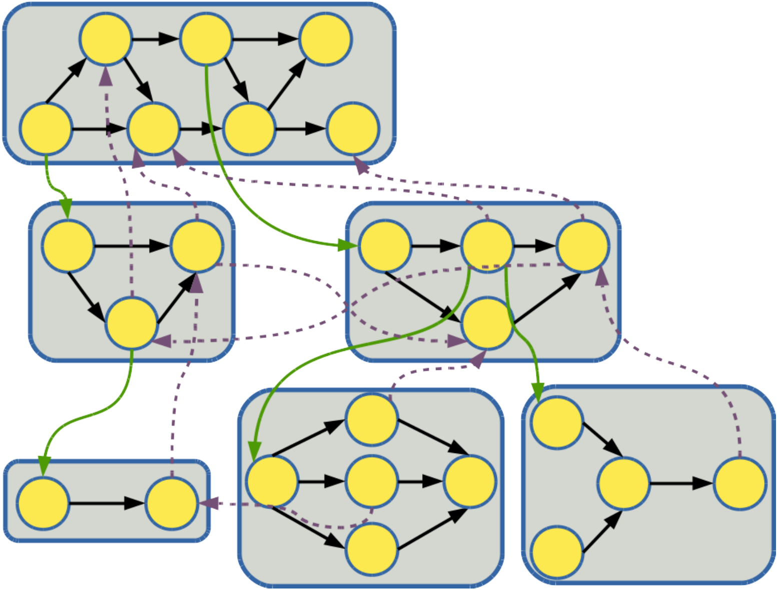
Fibers Firing Rule

- A fiber becomes enabled if it has received all input signals
- An enabled fiber may be selected for execution when the required hardware resource has been allocated
- When a fiber finishes its execution, a signal is sent to all destination threads to update the corresponding synchronization slots

Fibers States

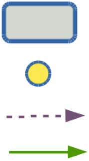


The EARTH Model of Computation



The EARTH Multithreaded Execution Model

Two levels of multithreading:
- Threaded procedures
- Fibers

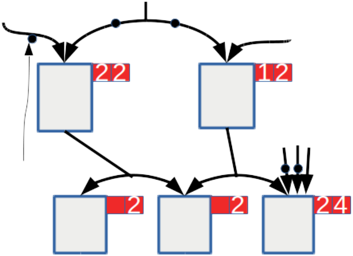


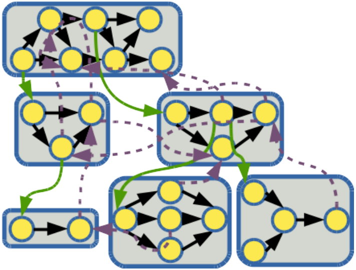
Async. function invocation

Fiber within frame

SYNC op.

Invoke threaded function





EARTH vs. Cilk

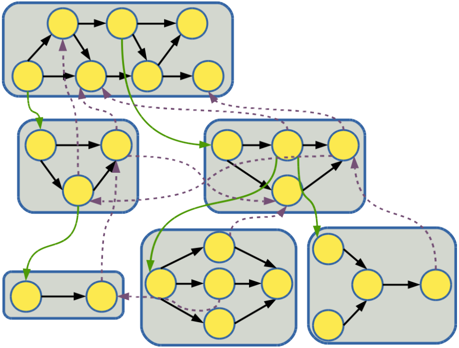


Figure: EARTH Model

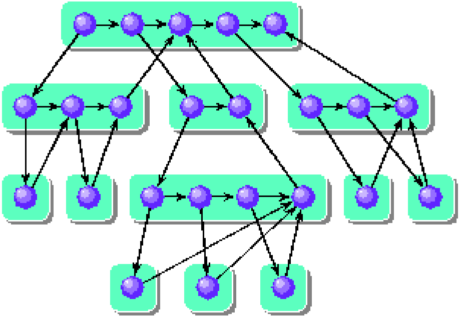
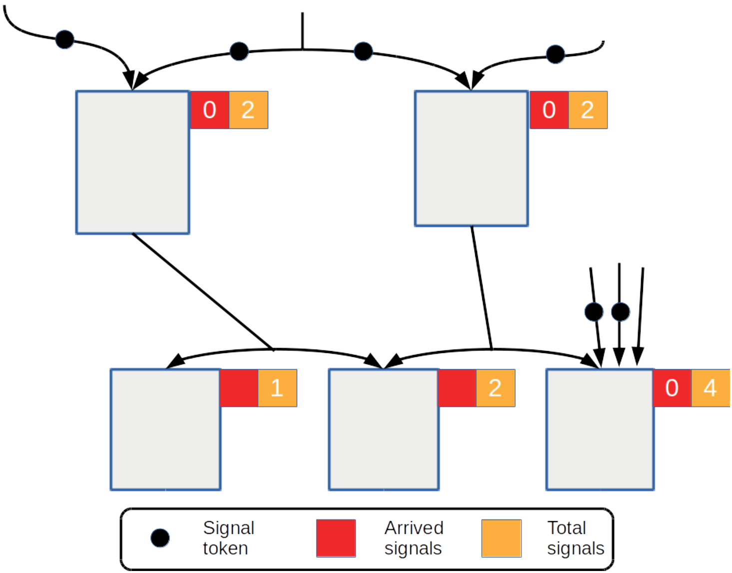
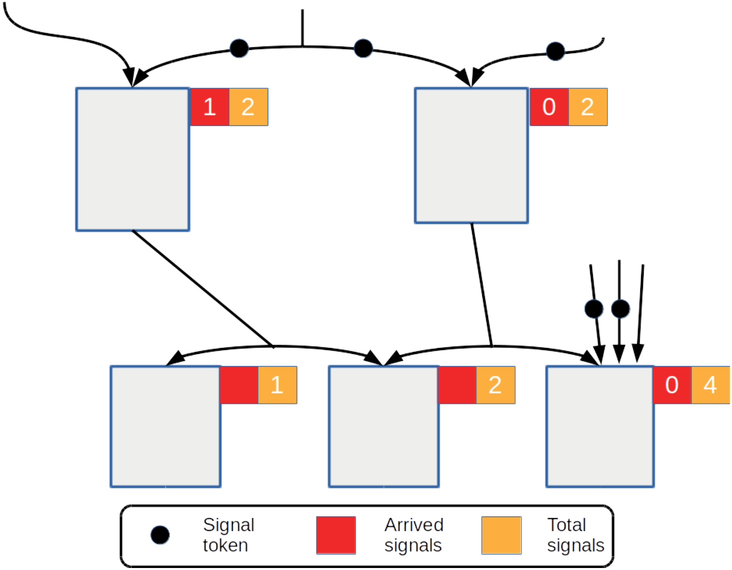


Figure: Cilk Model

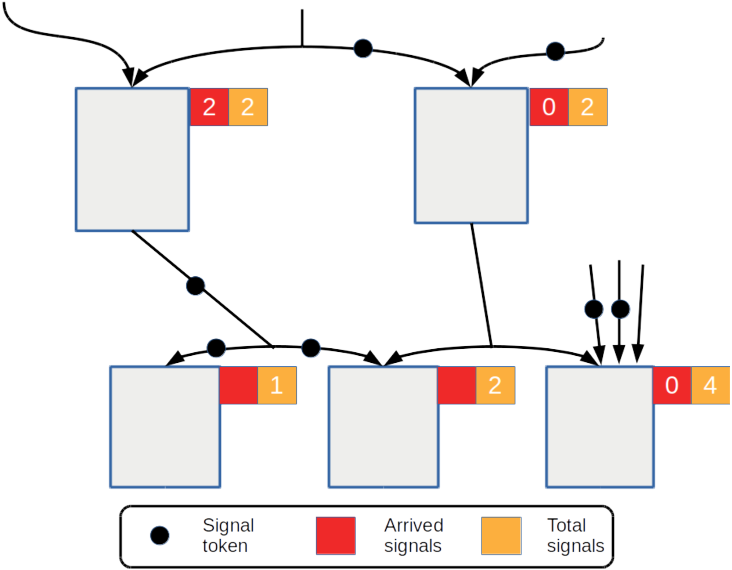
The Fiber Execution Model



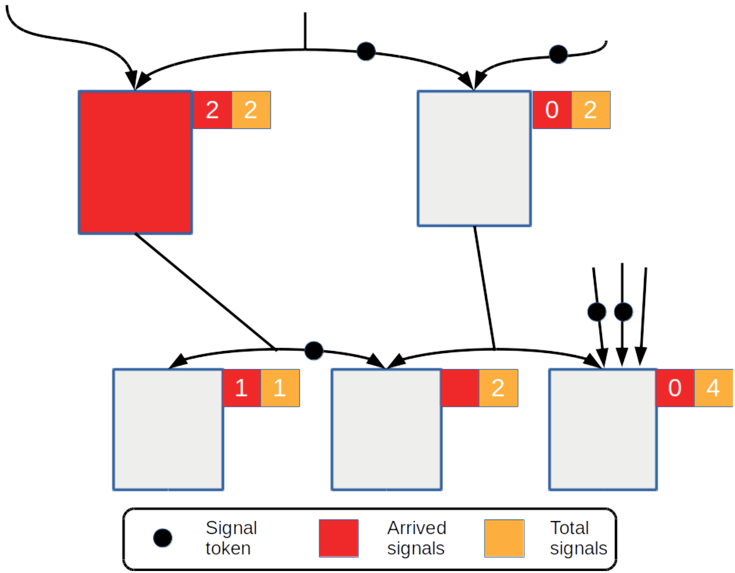
The Fiber Execution Model



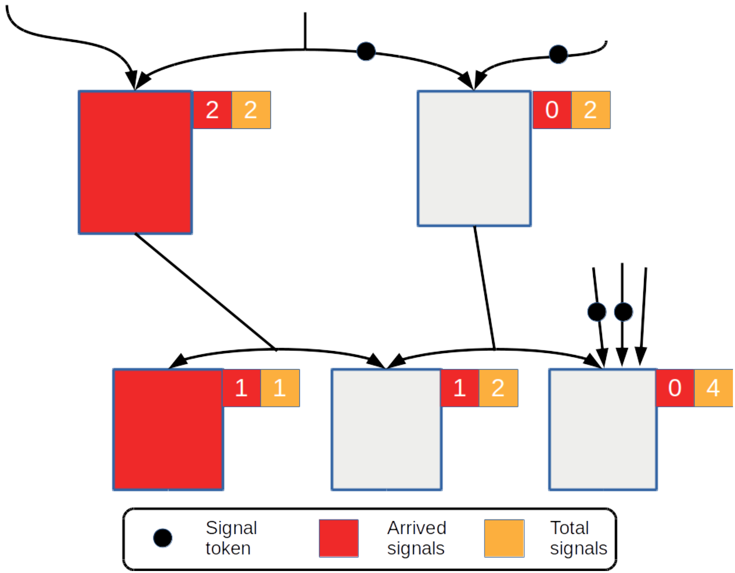
The Fiber Execution Model



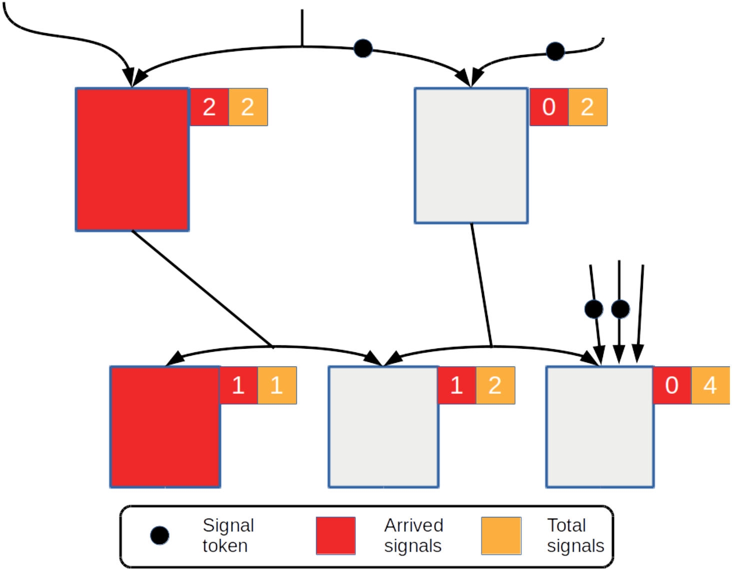
The Fiber Execution Model



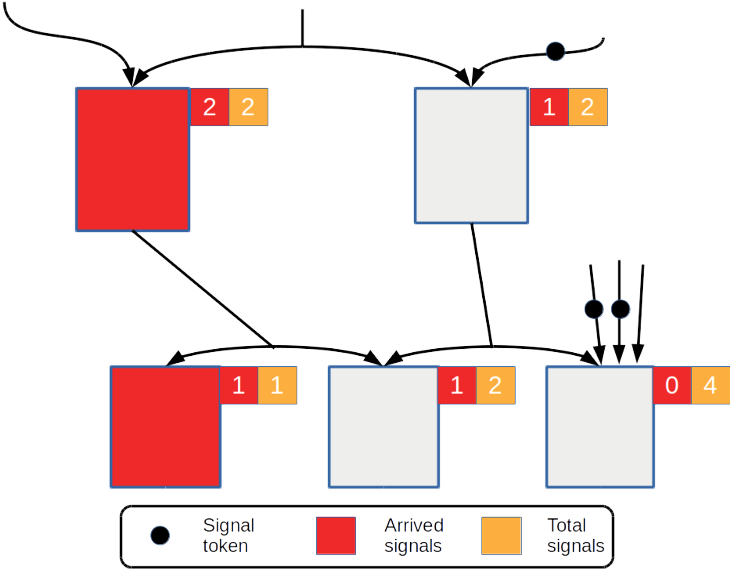
The Fiber Execution Model



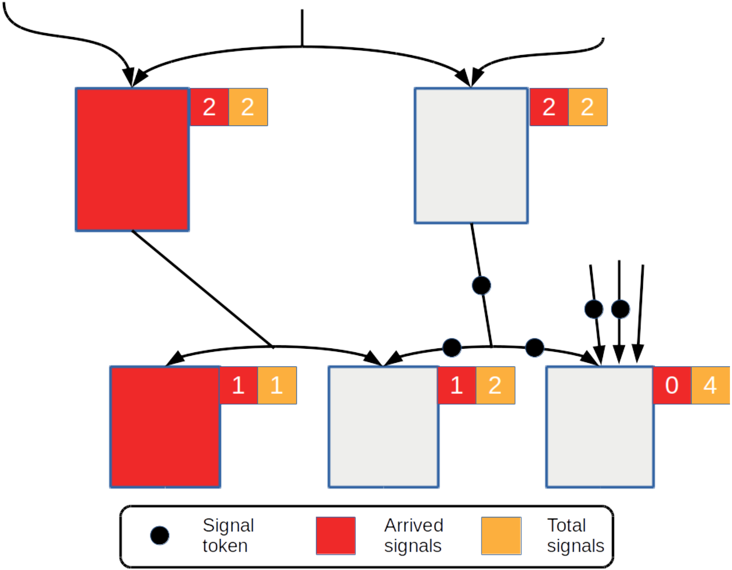
The Fiber Execution Model



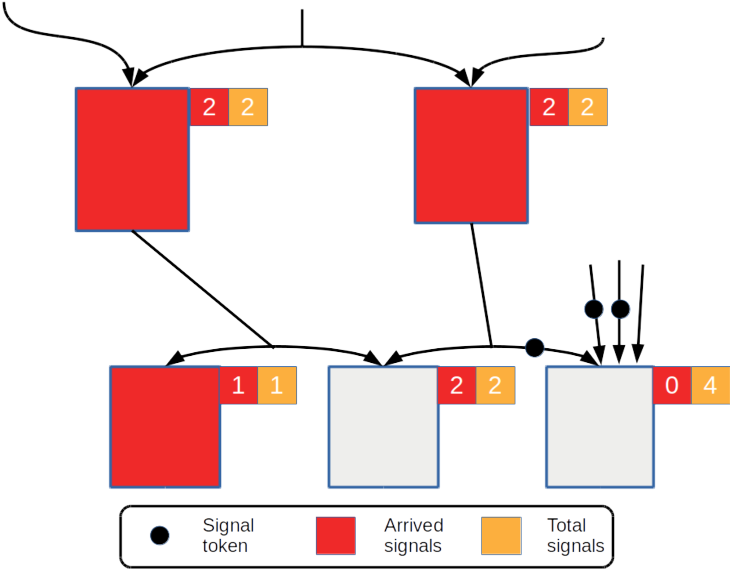
The Fiber Execution Model



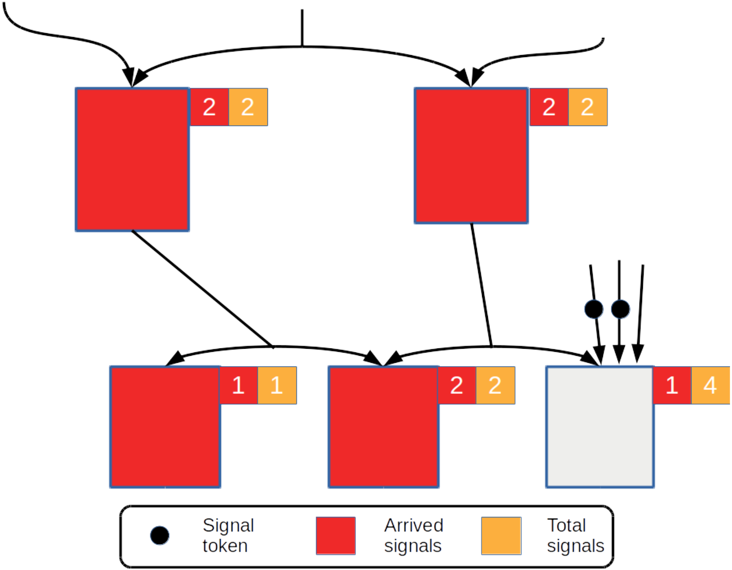
The Fiber Execution Model



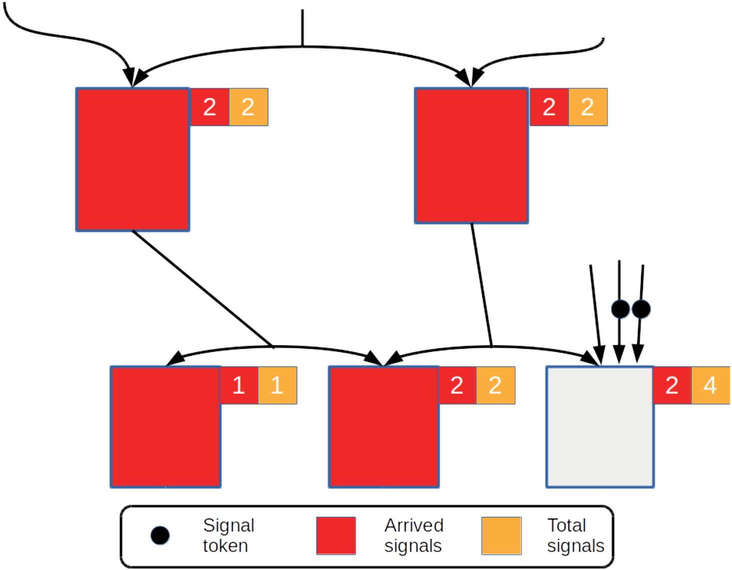
The Fiber Execution Model



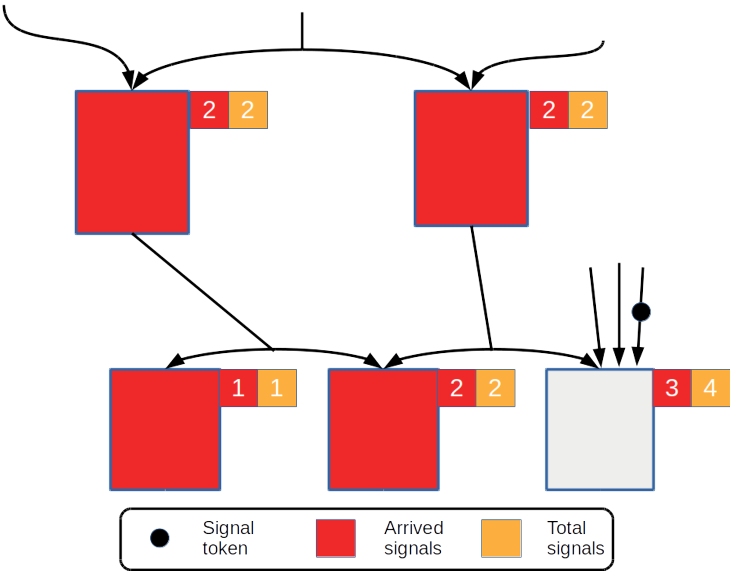
The Fiber Execution Model



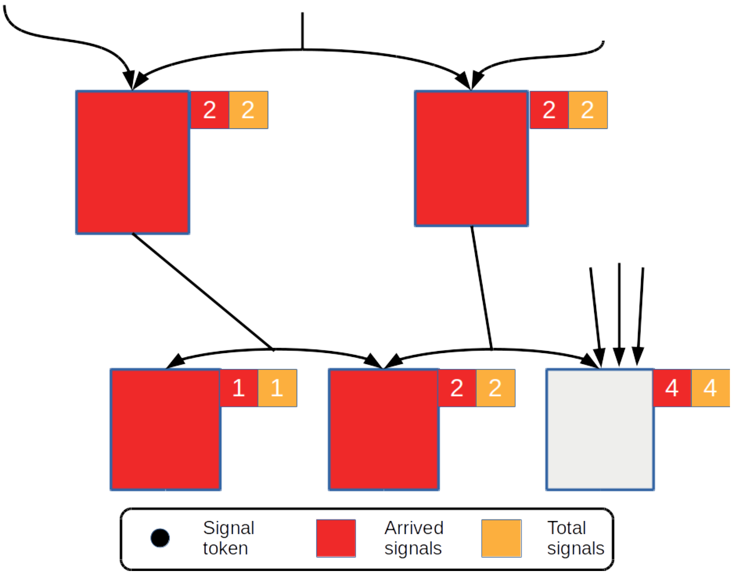
The Fiber Execution Model



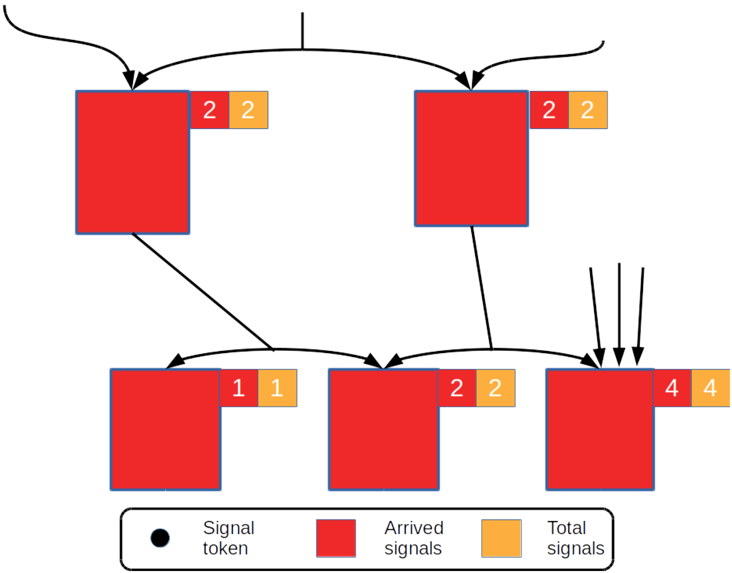
The Fiber Execution Model



The Fiber Execution Model

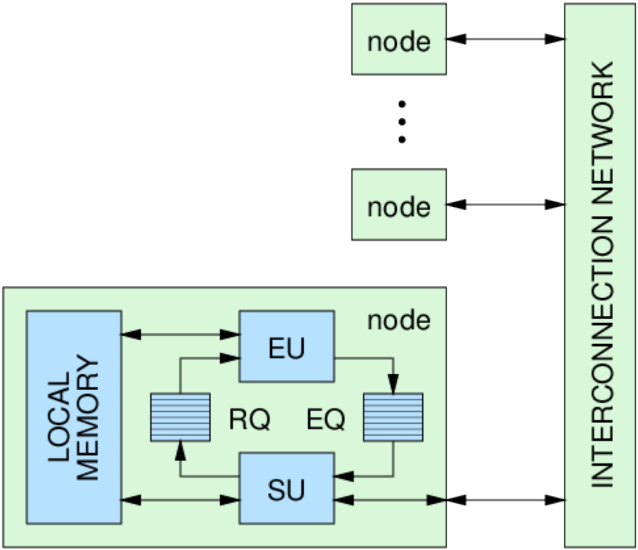


The Fiber Execution Model

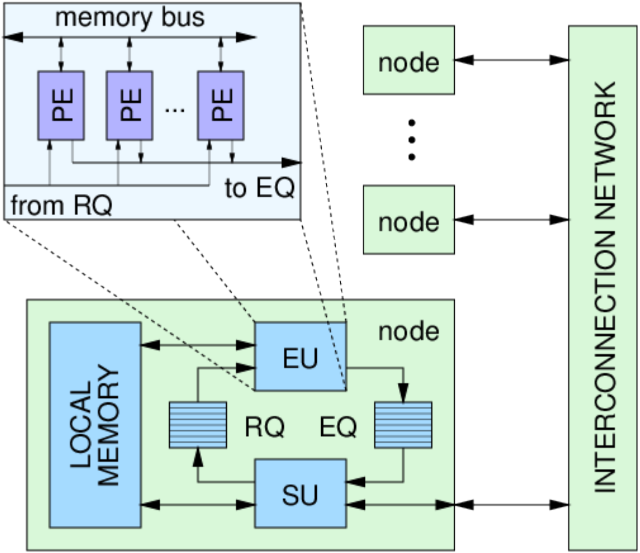


Outline

The EARTH Abstract Machine



The EARTH Abstract Machine



EARTH's Evaluation Platforms

EARTH-Manna

Implement EARTH on a bare metal tightly coupled multi-processor

EARTH's Evaluation Platforms

EARTH-Manna

Implement EARTH on a bare metal tightly coupled multi-processor

EARTH-IBM-SP

Plan to implement EARTH on an off-the-shelf commercial parallel machine (IBM SP2/SP3)

EARTH's Evaluation Platforms

EARTH-Manna

Implement EARTH on a bare metal tightly coupled multi-processor

EARTH-IBM-SP

Plan to implement EARTH on an off-the-shelf commercial parallel machine (IBM SP2/SP3)

EARTH on Clusters

- EARTH on Beowulf
- Implement EARTH on a cluster of UltraSPARC SMP workstations connected by Fast Ethernet

EARTH's Evaluation Platforms

EARTH-Manna

Implement EARTH on a bare metal tightly coupled multi-processor

EARTH on Clusters

- EARTH on Beowulf
- Implement EARTH on a cluster of UltraSPARC SMP workstations connected by Fast Ethernet

EARTH-IBM-SP

Plan to implement EARTH on an off-the-shelf commercial parallel machine (IBM SP2/SP3)

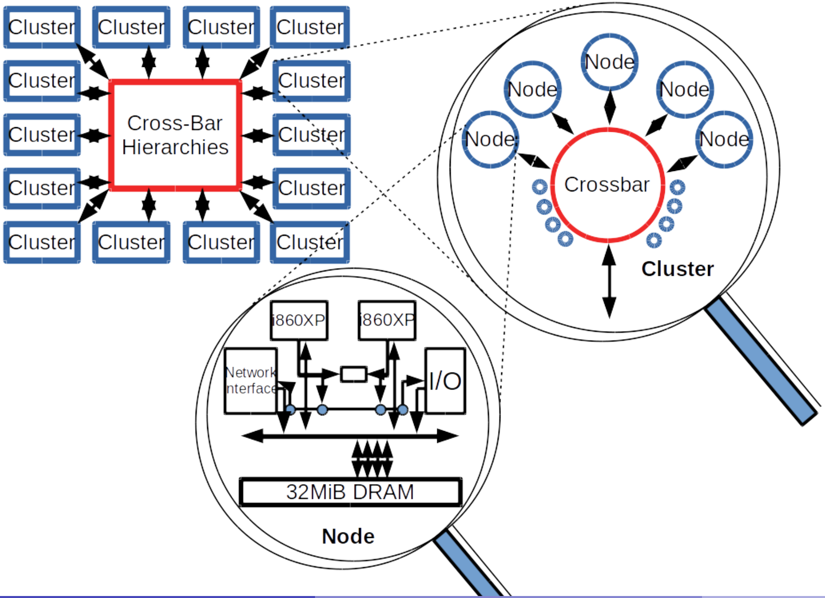
Note—Benchmark codes were all written using EARTH Threaded-C: The API for EARTH Execution and Abstract Machine Models

Outline

Open Issues

- Can a multithreaded program execution model support high scalability for large-scale parallel computing while maintaining high processing efficiency?
- If so, can this be achieved without exotic hardware support?
- Can these open issues be addressed both qualitatively and quantitatively with performance studies of real-life benchmarks?

The EARTH-MANNA Multiprocessor Testbed



Main Features of EARTH Multiprocessor

- Fast thread context switching
- Efficient parallel function invocation
- Good support of fine-grain dynamic load-balancing
- Efficient support of split-phase transactions
- Brings together the concept of fibers and dataflow

Main Features of EARTH Multiprocessor

- Fast thread context switching
- Efficient parallel function invocation
- Good support of fine-grain dynamic load-balancing
- Efficient support of split-phase transactions
- Brings together the concept of fibers and dataflow
- All that using off-the-shelf microprocessors!

EARTH-C Compiler Environment

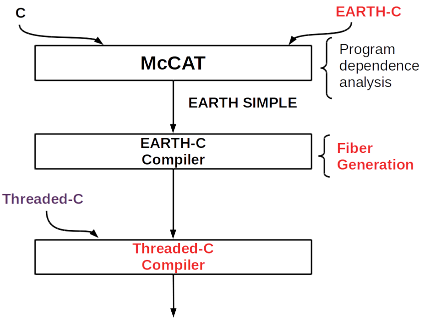


Figure: EARTH Compilation Environment

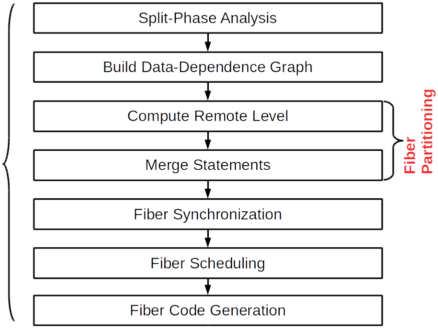


Figure: EARTH-C Compiler

- Overview
- Microbenchmarking:
  - Stress-testing
  - Measure performance of basic EARTH mechanisms for communication and synchronization
- Kernel benchmarking:
  - Speedup
  - USE value (Uni-node Support Efficiency)
    - *i.e.*, compare pure sequential kernel vs. single-node EARTH kernel
  - Latency tolerance capacity

Note—It is important to define your own performance "features" and/or "parameters" that best distinguishes your model from your competitors

EARTH Benchmark Suite (EBS)

Benchmark Name	Problem Size	Problem Domain	Characteristics
Ray Tracing	512×512	Image Processing	Class A
Wave-2D	150×150	Fluid Dynamics	Class A
Tomcatv	257	Scientific computation	Class A
2D-SLT	80×80	Fluid Dynamics	Class A
Matrix Multiply	480×480	Numerical computation	Class A
Barnes-Hut	8192 bodies	N-Body simulation	Class B
MP3D	18 K particles	Fluid Flow simulation	Class B
EM3D	20 K nodes	Electromagnetic wave simulation	Class B
Sampling Sorting	64 K	Sorting problem	Class B
Gauss Elimination	720×720	Numerical computation	Class B
Protein Folding	$3 \times 3 \times 3$ cube	Chemistry	Class B
Eigenvalue	2999	Numerical computation	Class B
Vertex Enumeration	10	Pivot-based searching	Class B
TSP	10	Graph searching	Class B
Paraffins	20	Chemistry	Class B
N-Queen	12	Graph searching	Class B
Power	10000	Power system optimization	Class B
Voronoi	64 K	Graph Partitioning	Class B
Heuristic-TSP	32 K	Searching problem	Class B
Tree-Add	1 M	Graph Searching	Class B

Highlighted benchmarks are implemented using a portable version of Threaded-C

- Efficient multithreading support is possible with off-the-shelf processor nodes with low overhead
  - At the time: context-switch time ≈ 35 cycles
  - Nowadays, this figure would most likely be bigger by at least one order of magnitude, probably ≈ 20 times bigger
    - This is speculation, but even with a bare-metal implementation, there is a ≈ 3× difference between memory bus and processor frequencies.
- A multithreaded program execution model an make a big difference
  - Results from the EARTH Benchmark Suite (EBS)

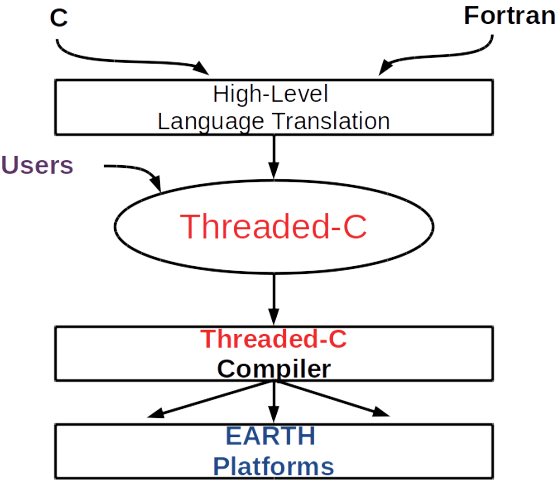
Outline

Outline

Threaded-C: A Base Language

- Serves as a target language for high-level language compilers
- Serves as a machine language for the EARTH architecture

The Role of Threaded-C



Outline

Features of Threaded Programs

- Thread partition
  - Thread length vs. useful parallelism
  - Where to “cut” a dependence and create a “split-phase?”
- Split-phase synchronization and communication
- Parallel threaded function invocation
- Dynamic load-balancing
- Other advanced features: fibers and dataflow

Base operations

- Thread synchronization and scheduling ops
  - SPAWN
  - SYNC
- Split-phase data & synchronization operations
  - GET_SYNC
  - DATA_SYNC
- Threaded function invocation and load-balancing operations
  - INVOKE
  - TOKEN

The EARTH Instruction Set I

- Basic insructions:
  - Arithmetic, logic, branching
  - Typical RISC instructions, *e.g.*, those from the i860
- Thread Switching
  - `FETCH_NEXT`
- Synchronization
  - `SPAWN fp, ip`
  - `SYNC fp, ss_off`
  - `INIT_SYNC ss_off, sync_cnt, reset_cnt, ip`
  - `INCR_SYNC fp, ss_off, value`
- Data transfers & synchronization
  - `DATA_SPAWN value, dest_addr, fp, ip`
  - `DATA_SYNC value, dest_addr, fp, ss_off`
  - `BLOCKDATA_SPAWN src_addr, dst_addr, size, fp, ip`
  - `BLOCKDATA_SYNC src_addr, dst_addr, size, fp, ss_off`

The EARTH Instruction Set II

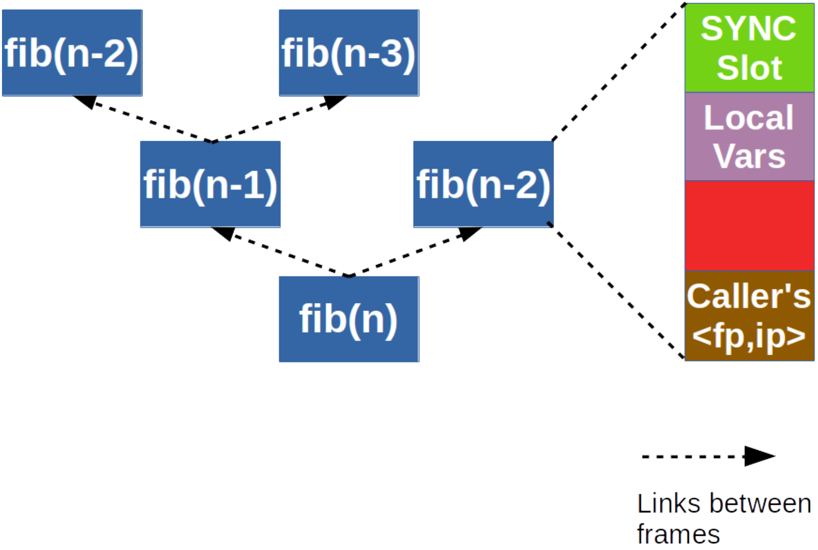
- Split-phase data requests
  - GET_SPAWN src_addr, dst_addr, fp, ip
  - GET_SYNC src_addr, dst_addr, fp, ss_off
  - GET_BLOCK_SPAWN src_addr, dst_addr, fp, ip
  - GET_BLOCK_SYNC src_addr, dst_addr, fp, ss_off
- Function invocation
  - INVOKE dst_PE, func_name, num_params, params
  - TOKEN func_name, num_params, params
  - END_FUNCTION

Outline

- **Ray Tracing** is a program for rendering 3-D photo-realistic images
- **Protein Folding** is an application that computes all possible foldings tructures of a given polymer
- **TSP** is an application to find a minimal-length Hamiltonian cycle in a graph with N cities and weighted paths.
- **Tomcatv** is one of the SPEC benchmarks which operates upon a mesh
- **Parrafins** is another application which enumerates distinct isomers paraffins
- **2D-SLT** is a program implementing the 2D-SLT Semi-Lagrangian Advection Model on a Gaussian Grid for numerical weather predication
- **N-Queens** is a benchmark program typical of graph searching problem.

Outline

Reminder: Tree of Activation Frames



The Fibonacci Example

Definition: Fibonacci Sequence

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

```
u64  fib(u64 n) {
    if (n < 2) return n;
    return fib(n-1) + fib(n-2);
}
```

The Fibonacci Example
Threaded-C Version

```
THREADED fib(u64 n, u64* result, sslot_t done) {
THREAD-0:
    u64 sum1 = 0, sum2 = 1;
    sslot_t slot1 = ... , /*,  , (0,0),  , */,
            slot2 = ... , /*,  , (2,2),  , */;
    if (n < 2) {
        DATA_RSYNC(n, result, done);
    } else {
        TOKEN (fib, n-1, &sum1, slot1);
        TOKEN (fib, n-2, &sum2, slot2);
    }
    END_THREAD();
THREAD-1:
    DATA_RSYNC(sum1+sum2;, result, done);
    END_THREAD();

    END_FUNCTION;
}
```

Although the definition is general, we set our numbers in either $\mathbb{R}$ or $\mathbb{C}$.
We generalize with the set $\mathbb{K}$.
Let A, B, and C be matrices, with $A_{M,K}$, $B_{K,N}$, $C_{M,N} \in \mathbb{K} \times \mathbb{K}$; let α, $\beta \in \mathbb{K}$. Then,

$$C_{M,N} \leftarrow \beta \times C_{M,N} + \alpha \times A_{M,K} \times B_{K,N}$$

One element $c_{i,j}$ of $C_{M,N}$ is computed as such:

$$c_{i,j} = \beta \cdot c_{i,j} + \alpha \cdot \sum_{k=1}^{k=K} a_{i,k} \cdot b_{k,j},\ a_{i,k} \in A_{M,K},\ b_{k,j} \in B_{K,N}$$

GEneral Matrix Multiplication (GEMM)

Square Case — Naïve Sequential Code

Note—To simplify the problem, we assume $\beta = 0$, $\alpha = 1,0$. In other words, we compute $C_{N,N} = A_{N,N} \times B_{N,N}$.

```
void dgemm(const double  beta,          double* C,
           const double  alpha,
           const double* A,     const double* B,
           const size_t N)
{
    for (size_t i = 0; i < N; ++i) {
        for (size_t j = 0; j < N; ++j) {
            c[i*N+j] = beta * c[i*N+j];
            for (size_t k = 0; k < N; ++k)
              c[i*N+j] += alpha * A[i*N+k] * b[k*N+j];
        }
    }
}
```

The Fibonacci Example

Threaded-C Version — Outer Product

Note—We assume the B matrix is correctly transposed to allow row-major traversal.

```
THREADED void dgemm(double* C, const double* A, const double* B,
                    const size_t N)
{
    double *row_a, *column_b;
    sslot_t slot0 = init_sslot(0,0),
            slot1 = init_sslot(N*N,N*N);
THREAD-0:
    for (size_t i = 0; i < M; ++i) {
        for (size_t j = 0; j < N; ++j) {
            row_a    = a[i];
            column_b = b[j];
            for (size_t k = 0; k < K; ++k)
                TOKEN (inner, &c[i*N+j], row_a, column_b, slot2);
        }
    }
    END_THREAD();
THREAD-1:
    RETURN(); , //,  , do,  , nothing
    END_THREAD();

    END_FUNCTION;
}
```

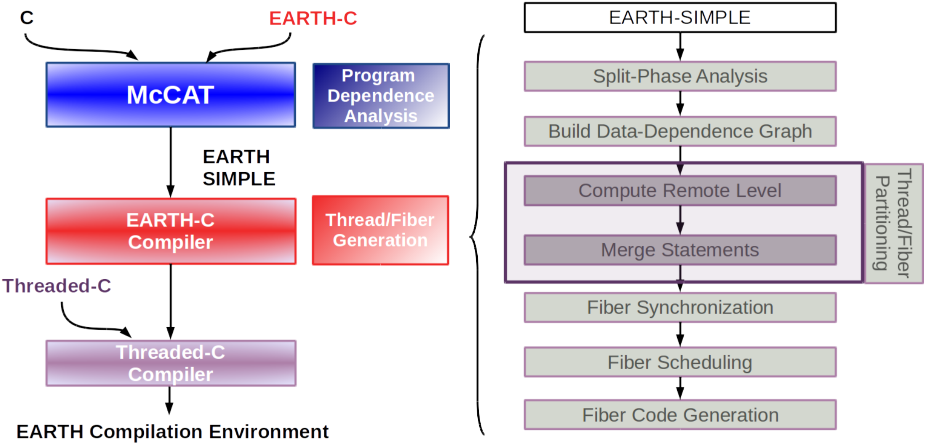
The Fibonacci Example

Threaded-C Version — Inner Product

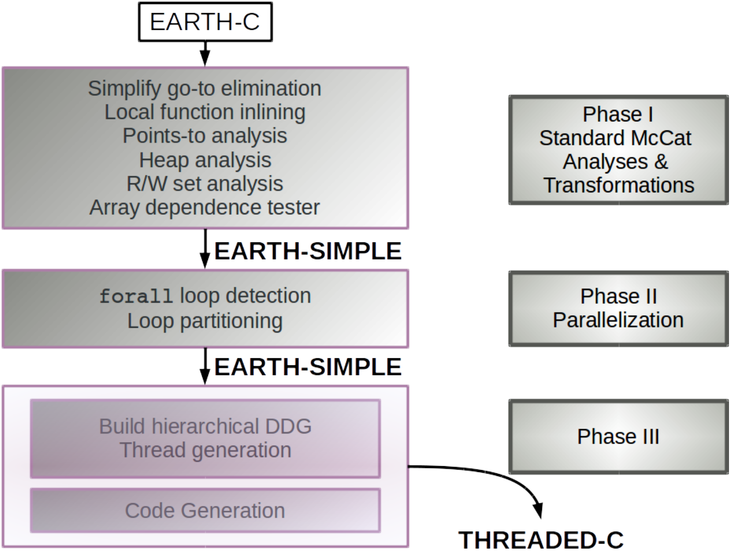
Note—We assume the B matrix is correctly transposed to allow row-major traversal.

```
THREADED void inner(       double* result, const size_t  N,
                     const double* A,      const double* B,
                     sslot_t       done)
{
    double sum, *row_a, *column_b;
    sslot_t slot0 = init_sslot(0,0),
            slot1 = init_sslot(2,2);
THREAD-0:
    BLKMOV_SYNC(A, row_a,    N, slot1);
    BLKMOV_SYNC(B, column_b, N, slot1);
    sum = 0.0;
    END_THREAD();
THREAD-1:
    for (size_t i = 0; i < N; ++i)
        sum += row_a[i] * column_b[j];

    DATA_RSYNC(*result +=sum;, done);

    END_THREAD();

    END_FUNCTION;
}
```

Compilation Environment, Revisited



Compilation Environment, Revisited



Main Features of EARTH

- ∆ Fast thread context switching
- Efficient parallel function invocation
- Good support of fine-grain dynamic load-balancing
- ∆ Efficient support of split-phase transactions and fibers

Note—Items marked with a ∆ are features unique to EARTH, and not found in the original Cilk model

Outline

Summary of EARTH-C

Extensions

- Explicit parallelism
  - Parallel vs. sequential statement sequences
  - `forall` loops
- Locality annotation
  - Local vs. remote memory references (global, local, replicate, ...)
- Dynamic load-balancing
  - Basic vs. remote function & invocation sites

References I

- Kevin Bryan Theobald (1999). "Earth: an efficient architecture for running threads". AAINQ50269. PhD thesis. Montreal, Que., Canada, Canada. ISBN: 0-612-50269-4
- Guang R. Gao and Vivek Sarkar (1995). "Location Consistency: Stepping Beyond the Memory Coherence Barrier". In: *ICPP (2)*, pp. 73–76
- Guang R. Gao and Vivek Sarkar (1997). "On the Importance of an End-To-End View of Memory Consistency in Future Computer Systems". In: *Proceedings of the International Symposium on High Performance Computing*. London, UK: Springer-Verlag, pp. 30–41. ISBN: 3-540-63766-4. URL: http://portal.acm.org/citation.cfm?id=646346.690059
- Guang R. Gao and Vivek Sarkar (2000). "Location Consistency-A New Memory Model and Cache Consistency Protocol". In: *IEEE Trans. Comput.* 49 (8), pp. 798–813. ISSN: 0018-9340. DOI: 10.1109/12.868026. URL: http://portal.acm.org/citation.cfm?id=354862.354865