Parallel Program Execution and Architecture Models With Dataflow Origin

-- The EARTH Experience

Guang R. Gao

ACM Fellow and IEEE Fellow
Endowed Distinguished Professor
Electrical & Computer Engineering
University of Delaware

ggao@capsl.udel.edu
Outline

• Parallel program execution models
• An evolution of dataflow architectures: experience with the argument-fetch dataflow model/architectures
• Evolution of fine-grain multithreaded program execution models – The EARTH experience.
• Memory and synchronization. models
• From EARTH to Runnemede – A Journey to extreme-scale
Outline

• Part I: EARTH execution model
• Part II: EARTH architecture model and platforms
• Part III: EARTH programming models and compilation techniques
• The percolation model and its applications
• Summary
Part I

EARTH: An Efficient Architecture for Running Threads

[PACT95, EURO-PAR95, ICS95, MASCOTS96, ISCA96, PACT96, PPoPP97, PACT97, SPAA97, DIPES98, SPAA98 and many others …]
The EARTH Program Execution Model

• What is a thread?
• How the state of a thread is represented?
• How a thread is enabled?
What is a Thread?

• A parallel function invocation
  *(threaded function invocation)*

• A code sequence defined *(by a user or a compiler)* to be a thread *(fiber)*

• Usually, a body of a threaded function may be partitioned into several threads
How to Execute Fibonacci Function in Parallel?

fib(4) ← fib(3) + fib(2)
   /         |
  /           |
fib(2)       fib(1)     fib(1)     fib(0)
   /         |
  /           |
fib(1)       fib(0)
Parallel Function Invocation

Tree of “Activation Frames”
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);
}
The Example is Partitioned into Four Fibers (Threads)

Thread0:

\[
\begin{align*}
a &= x[i]; \\
fact &= 1;
\end{align*}
\]

Thread1:

\[
\begin{align*}
fact &= fact \times a; \\
b &= x[j];
\end{align*}
\]

Thread2:

\[
\begin{align*}
sum &= a + b; \\
prod &= a \times b; \\
r1 &= g(sum); \\
r2 &= g(prod); \\
r3 &= g(fact);
\end{align*}
\]

Thread3:

\[
\text{return } (r1 + r2 + r3);
\]
The State of a Fiber (Thread)

• 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-light weighted”: it does not need dynamic storage (frame) allocation.

• Our focus: non-preemptive threads – called fibers
The “EARTH” Execution Model

“signal token”

a “thread” actor
The EARTH Fiber 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.
Thread States
The EARTH Model of Computation
The EARTH Multithreaded Execution Model

Two Level of Fine-Grain Threads:
- threaded procedures
- fibers

fiber within a frame
Async. function invocation
A sync operation
Invoke a threaded func
EARTH vs. CILK

Note: EARTH has its origin in static dataflow model
The “Fiber” Execution Model
The “Fiber” Execution Model
The “Fiber” Execution Model

- Signal Token
- Total # signals
- Arrived # signals
The “Fiber” Execution Model
The “Fiber” Execution Model

Diagram showing signal tokens and total and arrived signals.

Signal Token
Total # signals
Arrived # signals

Topic-C-EARTH 22
The “Fiber” Execution Model
The “Fiber” Execution Model
The “Fiber” Execution Model

![Diagram of the "Fiber" Execution Model]

- Signal Token
- Total # signals
- Arrived # signals
The “Fiber” Execution Model

Diagram showing the flow of signal tokens with nodes indicating the number of signals.
The “Fiber” Execution Model
The “Fiber” Execution Model
The “Fiber” Execution Model

![Diagram](image-url)
Part II

The EARTH

Abstract Machine (Architecture) Model

and

EARTH Evaluation Platforms
Execution Model and Abstract Machines

Programming Models → Execution Model → Execution Model API → Abstract Machine

Programming Environment Platforms

Users → Users
Execution Model and Abstract Machines
The EARTH Abstract Architecture (Model)
How To Evaluate EARTH Execution and Abstract Machine Model?
EARTH Evaluation Platforms

**EARTH-MANNA**
Implement EARTH on a *bare-metal* tightly-coupled multiprocessor.

**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

**NOTE:** Benchmark code are all written with EARTH Threaded-C: The API for EARTH Execution and Abstract Machine Models
EARTH-MANNA:
An Implementation of
The EARTH Architecture Model
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 (both Class A & B)?
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 split-phase transaction
- The concept of fibers and dataflow
EARTH-C Compiler Environment

(a) EARTH Compilation Environment

(b) EARTH-C Compiler
Performance Study of EARTH

• Overview
• Performance of basic EARTH primitives ("Stress Test" via "micro-benchmarks")
• Performance of benchmark programs
  – Speedup
  – USE value
  – Latency Tolerance Capacity

NOTE: It is important to design your own performance “features” or “parameters” that best distinguishes your models from your counterparts
# EARTH Benchmark Suite (EBS)

<table>
<thead>
<tr>
<th>Benchmark Name</th>
<th>Problem Size</th>
<th>Problem Domain</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ray Tracing</td>
<td>512 x 512</td>
<td>Image Processing</td>
<td>Class A</td>
</tr>
<tr>
<td>Wave-2D</td>
<td>150 x 150</td>
<td>Fluid Dynamic Problem</td>
<td>Class A</td>
</tr>
<tr>
<td>Tomcatv</td>
<td>257</td>
<td>Scientific Computation</td>
<td>Class A</td>
</tr>
<tr>
<td>2D-SLT</td>
<td>80 x 80</td>
<td>Fluid Dynamic Problem</td>
<td>Class A</td>
</tr>
<tr>
<td>Matrix Multiply</td>
<td>480 x 480</td>
<td>Numerical Computation</td>
<td>Class A</td>
</tr>
<tr>
<td>Barnes-Hut</td>
<td>8192 bodies</td>
<td>N-body Simulation</td>
<td>Class B</td>
</tr>
<tr>
<td>MP3D</td>
<td>18K particles</td>
<td>Fluid Flow Simulation</td>
<td>Class B</td>
</tr>
<tr>
<td>EM3D</td>
<td>20K nodes</td>
<td>Electromagnetic Wave Simulation</td>
<td>Class B</td>
</tr>
<tr>
<td>Sampling Sorting</td>
<td>64K</td>
<td>Sorting Problem</td>
<td>Class B</td>
</tr>
<tr>
<td>Gauss Elimination</td>
<td>720 x 720</td>
<td>Numerical Computation</td>
<td>Class B</td>
</tr>
<tr>
<td>Protein Folding</td>
<td>3 x 3 x 3 Cube</td>
<td>Chemistry</td>
<td>Class B</td>
</tr>
<tr>
<td>Eigenvalue</td>
<td>2999</td>
<td>Numerical Computation</td>
<td>Class B</td>
</tr>
<tr>
<td>Vertex Enumeration</td>
<td>10</td>
<td>Pivot-Based Searching</td>
<td>Class B</td>
</tr>
<tr>
<td>TSP</td>
<td>10</td>
<td>Graph Searching</td>
<td>Class B</td>
</tr>
<tr>
<td>Paraffins</td>
<td>20</td>
<td>Chemistry</td>
<td>Class B</td>
</tr>
<tr>
<td>N-Queen</td>
<td>12</td>
<td>Graph Searching</td>
<td>Class B</td>
</tr>
<tr>
<td>Power</td>
<td>10000</td>
<td>Power System Optimization</td>
<td>Class B</td>
</tr>
<tr>
<td>Voronoi</td>
<td>64K</td>
<td>Graph Partitioning</td>
<td>Class B</td>
</tr>
<tr>
<td>Heuristic-TSP</td>
<td>32K</td>
<td>Searching Problem</td>
<td>Class B</td>
</tr>
<tr>
<td>Tree-Add</td>
<td>1M</td>
<td>Graph Searching</td>
<td>Class B</td>
</tr>
</tbody>
</table>

- Portable Threaded-C exists
Main Experimental Results of EARTH-MANNA

• Efficient multithreading support is possible with off-the-shelf processor nodes with overhead
  – context switch time ~ 35 instruction cycles

• A Multithread program execution model can make a big difference
  – Results from the EARTH benchmark suit (EBS)
Part III

Programming Models for Multithreaded Architectures:

The EARTH Threaded-C Experience
Outline

• Features of multithreaded programming models
• EARTH instruction set
• EARTH benchmark suite (EBS)
• Programming examples
Threaded-C: A Base-Language

– To serve as a target language for high-level language compilers
– To serve as a machine language for EARTH architecture
The Role of Threaded-C

- C
- Fortran

High-level Language Translation

Threaded-C

Threaded-C Compiler

EARTH Platforms

Users
Features of Threaded Programming

- **Thread partition**
  - Thread length vs useful parallelism
  - Where to “cut” a dependence and make it “split-phase”? 
- **Split-phase synchronization and communication**
- **Parallel threaded function invocation**
- **Dynamic load balancing**
- **Other advanced features: fibers and dataflow**
The EARTH Operation Set

- The base operations
- Thread synchronization and scheduling ops
  - SPAWN, SYNC
- Split-phase data & sync ops
  - GET_SYNC, DATA_SYNC
- Threaded function invocation and load balancing ops
  - INVOKE, TOKEN
### Table 1. EARTH Instruction Set

- **Basic instructions**
  - Arithmetic, Logic and 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
Table 1. EARTH Instruction Set

- **Data Transfer & Synchronization**
  - DATA_SPAWN: value, dest_addr, fp, ip
  - DATA_SYNC: value, dest_addr, fp, ss_off
  - BLOCKDATA_SPAWN: src_addr, dest_addr, size, fp, ip
  - BLOCKDATA_SYNC: src_addr, dest_addr, size, fp, ss_off

- **Split_phase Data Requests**
  - GET_SPAWN: src_addr, dest_addr, fp, ip
  - GET_SYNC: src_addr, dest_addr, fp, ss_off
  - GET_BLOCK_SPAWN: src_addr, dest_addr, size, fp, ip
  - GET_BLOCK_SYNC: src_addr, dest_addr, size, fp, ip

- **Function Invocation**
  - INVOKE: dest_PE, f_name, no_params, params
  - TOKEN: f_name, no_params, params
  - END_FUNCTION
EARTH-MANNA Benchmark Programs

- **Ray Tracing** is a program for rendering 3-D photo-realistic images.
- **Protein Folding** is an application that computes all possible folding structures 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.
- **Paraffins** 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.
Parallel Function Invocation

Tree of “Activation Frames”
The Fibonacci Example

If $n < 2$

\[
\text{DATA_RSYNC (1, result, done)}
\]

else

\[
\{
\quad \text{TOKEN (fib, } n-1, \ & \text{sum1}, \ \text{slot}_1); \\
\quad \text{TOKEN (fib, } n-2, \ & \text{sum2}, \ \text{slot}_2); \\
\}
\]

END_THREAD();

THREAD-1;

\[
\text{DATA_RSYNC (sum1 + sum2;, result, done)}
\]

END_THREAD();

END_FUNCTION
Matrix Multiplication

```c
void main ( )
{
    int i, j, k;
    float sum;

    for (i=0; i < N; i++)
        for (j=0; j < N ; j++)   {
            sum = 0;
            for (k=0; k < N; k++)
                sum = sum + a [i] [k] * b [k] [j];
            c [i] [j] = sum;
        }
}
```

Sequential Version
BLKMOV_SYNC (a, row_a, N, slot_1);
BLKMOV_SYNC (b, column_b, N, slot_1);
sum = 0;
END_THREAD;

THREAD-1;
    for (i=0; i<N; i++ )
    
        sum = sum + (row_a[i] * column_b[i]);
        DATA_RSYNC (sum + result, done);

END_THREAD ( ) ;

END_FUNCTION

The Inner Product Example
for (i=0; i<N; i++)
    for (j=0; j<N; j++) {
        row_a = a[i];
        column_b = b[j];
        TOKEN (inner, &c[I][j], row_a,
               column_b, slot_1);
    }
END_THREAD;

THREAD-1;
RETURN ( );
END-THREAD

The Matrix Multiplication Example
EARTH-C Compiler Environment

The EARTH Compiler

EARTH Compilation Environment

EARTH SIMPLE

Threaded-C

Thread Code Generation

Thread Scheduling

Thread Synchronization

Merge Statements

Compute Remote Level

Build DDG

Split Phase Analysis

Program Dependence Analysis

Thread Generation

EARTH-C Compiler

Threaded-C Compiler

Program Dependence Analysis

McCAT

C

EARTH-C

EARTH SIMPLE

Threaded-C

EARTH Compilation Environment

The EARTH Compiler
The McCAT/EARTH Compiler

EARTH-C

- Simplify `goto` elimination
- Local function inlining
- Points-to Analysis
- Heap Analysis
- R/W Set Analysis
- Array Dependence Tester

EARTH-SIMPLE-C

- Forall Loop Detection
- Loop Partitioning

EARTH-SIMPLE-C

- Build Hierarchical DDG
- Thread Generation

Code Generation

THREADED-C

---

PHASE I
(Standard McCAT Analyses & Transformations)

PHASE II
(Parallelization)

PHASE III
Advanced Features in Threaded-C Programming
Main Features of EARTH

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

*Features unique to the EARTH model in comparison to the CILK model*
Summary of EARTH-C Extensions

• Explicit Parallelism
  – Parallel versus Sequential statement sequences
  – For all loops

• Locality Annotation
  – Local versus Remote Memory references (global, local, replicate, …)

• Dynamic Load Balancing
  – Basic versus remote function and invocation sites
Percolation Model
under the DARPA HTMT Architecture Project

A User’s Perspective

Primary Execution Engine
- Prepare and percolate “parceled threads”
- Perform intelligent memory operations
- Global Memory Management

High Speed CPUs
- CRAM
- CPUs

SRAM
- SRAM
- S-PIM Engine

DRAM
- DRAM
- D-PIM Engine

Main M

Topic-C-EARTH
**The Percolation Model**

- **What is percolation?**
  - Dynamic, adaptive computation/data movement, migration, transformation in-place or on-the-fly to keep system resource usefully busy

- **Features of percolation**
  - Both data and thread may percolate
  - Computation reorganization and data layout reorganization
  - Asynchronous invocation

---

**An Example of percolation—Cannon’s Algorithm**

- Data layout reorganization during percolation
- Cannon’s nearest neighbor data transfer

---

**HTML-like Architectures**

- Level 0: fast cpu
- Level 1 PIM
- Level 2 PIM
- Level 3
Another View: Codelets

1993: EARTH and 1997: HTMT
Gao, Hum, Theobald

(courtesy: Jack Dennis, DF Workshop, Oct 10. 2011, Gavelston, Tx)

- Group Instructions and Data into Blocks
- Pre-Fetch Input Data
- Non-Pre-emptive Execution
- Store Results in Fresh Memory
- Completion Enables Successor Codelets
- Requires Dynamic Memory Management

Several Current Projects are Studying Variations on this Concept
The Codelet: A Fine-Grain Piece of Computing

Data Objects \{ \}

Codelet

Result Object

Supports Massively Parallel Computation!
The Codelet: A Fine-Grain Piece of Computing

This Looks Like Data Flow!!