Toward A Dynamic, Asynchronous and Adaptive Execution Model –
A Path to Runnemedede Codelet Model and Beyond

Guang R. Gao
Distinguished Professor, Dept. of ECE
University of Delaware
And
Founder, ET. International Inc.
Outline

• Introduction
• Fine-Grain Execution Models – A History Perspective
• Cyclops-64 Execution Model (TNT) and Bare-Metal OS Approach
• From TNT to Codelet: A Path to The UHPC/Runnemedede Execution Model
• SWARM - Preliminary Results (e.g. Graphs-500, etc.)
• Future Directions – Beyond Runnemedede
An Abstract Machine Model for Extreme Scale Systems
Outline

• Introduction
• Fine-Grain Execution Models – A History Perspective
• Cyclops-64 Execution Model (TNT) and Bare-Metal OS Approach
• From TNT to Codelet: A Path to The UHPC/Runnemedede Execution Model
• Preliminary Results (e.g. Graphs-500, etc.)
• Future Directions – Beyond Runnemedede
Terminology Clarification

• Parallel Model of Computation
  – Parallel Models for Algorithm Designers
  – Parallel Models for System Designers
• Parallel Programming Models
• Parallel Execution Models
• Parallel Architecture Models
What Does Program Execution Model (PXM) Mean?

- The notion of PXM

The program execution model (PXM) is the basic low-level abstraction of the underlying system architecture upon which our programming model, compilation strategy, runtime system, and other software components are developed.

- The PXM (and its API) serves as an interface between the architecture and the software.

- PXM and Abstract Machine
What is A Shared Memory Execution Model?

**Thread Model**
A set of rules for creating, destroying and managing threads

**Execution Model**

**Memory Model**
Dictate the ordering of memory operations

**Synchronization Model**
Provide a set of mechanisms to protect from data races

The Thread Virtual Machine
Case Studies of Fine-Gran Execution Models

- Dataflow Model (1970s - )
- EARTH Model (1988 - )
- HTVM Model (2000 - )
- Runnemedede Codelet Model (2010 - )
On The Weakness of The Pure Dataflow Model

• Overhead on instruction-level dataflow
• Abstract machine model is somewhat naïve
  – A shared storage/memory model with single-assignment semantics
  – Lacks an abstraction for locality
  – Lacks an abstraction for parallelism/resource scheduling and control
• Non-determinate computation is a challenge.
The Weakness of EARTH Model

• ??
Outline

• Introduction
• Fine-Grain Execution Models – A History Perspective
• Cyclops-64 Execution Model (TNT) and Bare-Metal OS Approach
• From TNT to Codelet: A Path to The UHPC/Runnede Execution Model
• SWARM - Preliminary Results (e.g. Graphs-500, etc.)
• Future Directions – Beyond Runnede
What is Cyclops64?

- Among the first general programmable large-scale many-core
- Intra- and Inter-node scalability by design
- 3D network mesh especially suited to physical problems
- Currently being deployed toward peta-scale levels
- Powered by ETI Software Solutions

11/1/2011
Cyclops-64 Architecture

- 160 Thread Units
- 64 General Purpose Registers
- ~4.6MB On-chip Memory
- 1 GB Off-chip Memory
Execution Model and Abstract Machines

Programming Environment Platforms

Execution Model API

Abstract Machine

Programming Models

Execution Model

Users
A Nice Execution Model for C64?

- Challenge of scaling up to 1,000,000 fold
- Scalable parallelism at two levels:
  - At the node level – 10,000+ nodes
  - At the chip level – 160 cores and memory banks per node
- Let us begin with a solid program execution model and its API
10,000 Feet High System Overview
- A *Heterogeneous* Architecture Model

Front-end
Programming environment

Interconnection network

Back-end
C64 computing *accelerator* engine

Courtesy from E.T. International Inc
Cyclops-64 Software Architecture

- Host network
- Front-end cluster
- Monitoring nodes
- Control network
- Gig Ethernet
- Login nodes
- Admin nodes
- Traditional OS
- TNT Thread Virtual Machine
- C64 computing engine
TNT (TiNy-Thread) – The API of The Execution Model

- Multi-chip multiprocessor extension of the base C64 ISA.
- Runs *directly* on top of C64 HW architecture.
- Takes advantage of C64 *HW features* to achieve high scalability.
- Three components: *thread model, memory model* and *synchronization model*.
Base Program Execution Model

- Propose TNT (TiNy-Thread) as a API of the C64 execution model (virtual C64 instruction set)
- Implement TNT from scratch – no burden from “porting” conventional OS!
- Take a library approach with pragma/directives – avoid a show-stopper due to lack of early optimizing compiler
- TNT API is evolved (from day one) with constant advice and feedback from a team of dedicated domain experts
- TNT grows from familiar PThread syntax: drastically reduces the early learning curve.
- TNT provides adequate (architect certified) performance feedback for both high-level programming models as well as compiler technology R&D
Cyclops-64 Base Thread Virtual Machine – At MTVM/LGT Level

- C64 TVM API
- TiNy Threads
- tnt_create(..), tnt_kill(..),
  tnt_suspend(..),
  tnt_awake(..)
- Thread management:
  create, terminate,
  suspend, awake
- Mutual exclusion,
  direct thread-to-thread,
  barrier synchronization
- tnt_load(..),
  tnt_getdata_sync(..),
  tnt_putdata_sync(..)
- Global address space,
  SW controlled cache
- tnt_mutex_lock(..),
  tnt_signal(..), tnt_wait(..),
  tnt_barrier(..)
C64 Programming Development Package
(with **TNT-C** programming interface)

**Front-End System**
- Boot-Up Software
- Job Scheduler
- GDB Debugger
- Performance Tools

**Multi-Core Chip (Node)**
- Core 1
- Core 2
- ... (Core N)
- Global Memory

**Parallel APIs (SHMEM, etc.)**
- Newlib
- TNT Library
- Inter-chip Drivers

**ETI Compiler**
- Binutils (AS, LD)

**Customer Applications**
- ETI Multi-threaded, Multi-node Benchmarks

**Example Applications:**
- LU NPB
- MXM
- 1D/2D FFT
- Mandelbrot Set
- Graphics Rendering
- Many more...
DEEP - Delaware End-To-End Emulation Engine

- Gate-level emulation of up to 20 C64 chips
- System software and gate-level architecture are co-validated
- 10,000 programs running, many are TNT code
- Within 2 month: 100+ billion C64 cycles are emulated
- allow a considerable number of real life parallel programs be successfully executed and tested.
What We Learned With TNT?

• A robust C64 programming environment is deployed long before the hardware is available (TNT as a target for C64 SHMEM, OpenMP)

• ~15,000 TNT test programs are deployed on C64 - a critical mass for experimentation

• Enable important compiler optimization investigation through detailed TNT studies
A Report on Early Experience on C64

• Case Study I: Programming Kernels:
  – Monte-Carlo
  – Matrix Multiply
  – FFT
  – LU Decomposition
  – Dynamic programming
  – SCCA2
  – othes

• Case Study II: Mapping OpenMP and SHMEM on C64

• Other Codes
Performance with Optimizations – LU
(a good list of compiler optimizations to study)

Performance with Optimizations (156 TUs)

Execution of 1000x1000 LU in SRAM

1 – Base Parallel Version
2 – + Dyn. Repartitioning + Recursion
3 – + Processor Adaptation
4 – + Hardware Barrier
5 – + Reg. Tiling (Manual)
6 – + Instruction Scheduling (Manual)

27.5 GFLPS
Performance of SCCA2 (kernel 4)

- **scalability**
  - Scale well with # cores
  - Linear speedup for #cores < 32
- **incremental improvement**
  - Competitive vs. MTA-2

<table>
<thead>
<tr>
<th>#threads</th>
<th>C64</th>
<th>SMPs</th>
<th>MTA2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2917082</td>
<td>5369740</td>
<td>752256</td>
</tr>
<tr>
<td>8</td>
<td>5513257</td>
<td>2141457</td>
<td>619357</td>
</tr>
<tr>
<td>16</td>
<td>9799661</td>
<td>915617</td>
<td>488894</td>
</tr>
<tr>
<td>32</td>
<td>17349325</td>
<td>362390</td>
<td>482681</td>
</tr>
</tbody>
</table>

Unit: TEPS -- Traversed Edges per second

SMPs: 4-way Xeon dual-core, 2MB L2 Cache

[PPoPP2008, LCPC2008]
7 Year and $20+ M investment

- 7 year old expert engineering team of 25 people with 20% PhDs
- Cohesive and seasoned team with a track record of delivering on time and under budget
- Technology investment of $20+M and many FTE years
- Representing over 5.5M lines of deployed code
- Exclusive creator of bare-metal OS for Cyclops-64 Supercomputer consisting of upto 10,000 chips representing 1.6+ million CPUS cores
- Patent portfolio and IP assets
Cyclops – An Excellent *Sandbox* for Future Extreme-Scale System Studies

- The only *working peta-scale systems* with general programmable many-core chip technology
- The Cyclops system software stack is directly based on a solid parallel execution model and *does not carry the baggage of traditional OS noises*.
- It is robust and maintainable, and can carry *industrial-strength productive workloads* – avoiding the limitations of the usual university student software.
Cyclops Technology and Performance Evolution

Processor Technology Improvement

Current Cyclops Performance
90nm, 500MHz

New Generation Cyclops
32nm, 1GHz

Upgrade Host interface

Host Interconnect Bandwidth

Cyclops and Host Interconnect

FLOPS

16x

PCIe

4x
Cyclops Software Enhancement and Optimization

• Upgrading Cyclops system software with SWARM technology may have a great potential to improve critical production applications of the sponsors.

• A 2x to 10x improvement is possible without upgrading the existing hardware!

• And >> 10x is entirely possible for graph problems!
Outline

- Introduction
- Fine-Grain Execution Models – A History Perspective
- Cyclops-64 Execution Model (TNT) and Bare-Metal OS Approach
- From TNT to Codelet: A Path to The UHPC/Runnemedede Execution Model
- SWARM - Preliminary Results (e.g. Graphs-500, etc.)
- Future Directions – Beyond Runnemedede
The Concept of A Codelet
(DeepDive - Sept 14, 2009, modified)

- A codelet is a code unit that, when scheduled for execution (on a codelet execution unit) – the execution will not interfere with the “outside world” during its execution process.

- In other word, a codelet interacts with the “outside world” as if only through its inputs and outputs (happening at the beginning and the end of execution).

- Consequently, its scheduling is intrinsically non-preemptive
Concept of Codelet

(Feb. 4th, 2011)

- Codelets are the principal **scheduling quantum** under our codelet based execution model. As a principal scheduling quantum, a codelet once allocated and scheduled to a computation unit (e.g. a computation core), will be kept usefully busy - hence it will not be preempted *in most common cases.*

- The underline Runnemede hardware architecture and system software (e.g. compiler, etc.) are optimized to ensure such *non-preemption features can be productively utilized.*
Codelet Graphs


- Consequently, a unique feature of our CDG - it employs the “argument-fetching” dataflow principle [DennisGao88].

- From *data-driven* to *event-driven*
Clarification

• A codelet is **NOT** equivalent to a macro dataflow actor!
  – Event-driven vs. data-driven
  – Output condition is not “atomic” (important!)

• A codelet is **NOT** equivalent to a transaction in the transaction memory (TSM) model!
Well-Behaved Codelet Graphs (or Schemata)

**Concept:** A codelet graph $G$ is *well-behaved* if it becomes enabled iff each input has an event token. After the firing (execution), all input tokens should be consumed and a token will be generated on each output arc.

**NOTE:** A regular codelet is well-behaved
Examples: Ill-Behaved CDGs

Arbitrary connections of data flow operators can result in pathological programs, such as the following:

1. Deadlock
2. Hangup
3. Conflict
4. Unclean
Question: How To Ensure A CDG is Well-Behaved?

- **Method I:** After a CDG is constructed, check for well-behavedness. Then debug it.

- **Method II:** Provide a set (well-structured) construction rules. A CDG will be built by recursively applying these nice construction rules and producing a well-formed CDG.

**NOTE:** A Well-Formed Codelet Schema => Well-Behaved
Asynchronous Task Level Parallelism: Procedures and Loops (Work in Progress)

• Asynchronous procedure invocation: *event-driven futures* with *asynchronous split-phase call-backs* enabled by our base codelet model.

• Asynchronous loop nest unraveling: *event-driven software pipelining* with codelets

NOTE: both are leveraging on our EARTH model and loop nest software pipelining work
Another Two Key Questions Related to Memory Models

• Assuming two memory operations with the same destination memory location address X (i.e. LOAD X or STORE X) are issued through the same processing core.
  – Notes: We assume that the two memory operations are issued in their program order. Both of the two memory operations access memory location address X.
  – **Q2:** Should the hardware (architecture) permit > 1 alternative paths of routing of the memory operations (transactions) along the way?
  – **Q3:** If the answer of Q2 is true (I assume it is) - then it is possible that the two operations may arrive at their destination out-of-order?
Your Answers to the Questions?

**Q1**: Should the hardware (architecture) permit > 1 alternative paths of routing of the memory operations (transactions) along the way?

**Q2**: If the answer of Q1 is true (I assume it is) - then it is well possible that the two operations may arrive at their destination out-of-order?

<table>
<thead>
<tr>
<th>No.</th>
<th>Answer to Q1</th>
<th>Answer to Q2</th>
<th>Which one?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>
Possible Answers to the Questions

**Q1**: Should the hardware (architecture) permit > 1 alternative paths of routing of the memory operations (transactions) along the way?

**Q2**: If the answer of Q1 is true (I assume it is) - then it is possible that the two operations may arrive at their destination out-of-order?

<table>
<thead>
<tr>
<th>No.</th>
<th>Answer to Q1</th>
<th>Answer to Q2</th>
<th>Who answered</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>Yes</td>
<td>GG, MM</td>
</tr>
<tr>
<td>2</td>
<td>Yes</td>
<td>No</td>
<td>BS</td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>Yes</td>
<td>MS</td>
</tr>
<tr>
<td>4</td>
<td>No</td>
<td>No</td>
<td>DD, SS</td>
</tr>
</tbody>
</table>
One More Key Question on Memory Model?

• **Question Q4**: Should a memory model preserves the notion of *causality*?
Outline

• Introduction
• Fine-Grain Execution Models – A History Perspective
• Cyclops-64 Execution Model (TNT) and Bare-Metal OS Approach
• From TNT to Codelet: A Path to The UHPC/Runnemede Execution Model
• SWARM - Preliminary Results (e.g. Graphs-500, etc.)
• Future Directions – Beyond Runnemede
Our Experience

ETI
System Software and Dynamic Runtime Technology

Real Many-core Architectures

Intel X86 Supercomputer
- 12 cores/node
- 24 GB/node
- Infiniband network

Commercial Commodity Architecture

Intel UHPC-Runnemede
- 1024 cores/chip
- < 40 W/chip
- > 50 GF/W

Future Exascale Architecture

IBM Cyclops
- 160 cores/chip
- 1 GB/node
- 3D mesh network (24 GB/s)

IBM Cell
- 8 SPE + 1 PPC cores/chip

Intel Single Chip Cloud (SCC)
- 48 cores/chip
- 64 GB/node
- 2D on-chip interconnect

Adapteva Epiphany
- 64 cores/chip
- 70 GF/W
- 1024 cores/chip Q2/2012

Our Experience
What is SWARM?

- **SWift Adaptive Runtime Machine (SWARM):**
  - Runtime system for heterogeneous large-scale systems
  - Implements an execution model based on codelets

- **Goal:**
  - Pioneer a *production-quality industrial-strength software base for a dynamic, asynchronous and adaptive runtime*
Seminal Ideas of SWARM

**SWift Adaptive Runtime Machine (SWARM):**

- SWARM is based on a fine-grain, asynchronous, event-drive execution model leveraging 20+ year of founder’s R&D work and know hows on dataflow models and their extension - cumulated to the codelet model.
- SWARM expands the codelet model with major patent-pending extensions that allow smooth integration of realistic machine memory and scheduling constraints.
- SWARM technology features a patent-pending HW/SW co-design method that permits exploitation the tradeoffs of a combination of hardware and software implementation.
- SWARM incorporates resource, energy, resilience and security constraints into a unified event-driven execution model.
Hierarchical threading system
  Codelets: light-weight and run non-preemptively
  Event-driven
Resources are accessed through split-phase/non-blocking/asynchronous operation
Unique Features of ETI SWARM Technology

- It *hides* the hardware complexity and heterogeneity while providing an *intuitive, user-friendly interface* for applying *domain-specific* input (static and dynamic) to improve the program execution through *runtime optimization* under machine resource constraints.

- It supports a *self-aware* execution model that can dynamically and automatically adapt the program execution to best utilize the many-core hardware features and achieve users’ performance and energy goals.
Value Proposition

• SWARM can bring up to 10x (and beyond) in performance increase for hard-to-parallelize application on today’s multi-core platforms.

• SWARM has a great potential to achieve yet another significant performance increase when apply to future many-core chip platforms with low energy budget.

• SWARM achieve such big performance and efficient gain without excessive demand on user programming cost and knowledge of many-core complex hardware details.
ETI’s SWARM delivered the best multi-node results for every scale problem to which it was submitted.

ETI SWARM helps to bring 3 Intel platforms to the top 10 list:

- #6 - TACC's Lonestar running SWARM 4 delivered 8.1 GE/s at 512 nodes (Scale 34)
- #8 - Sandia National Lab's Red Sky running SWARM delivered 9.5 GE/s at 512 nodes (Scale 33)
- #9 - Intel's Endeavor running SWARM delivered 6.9 GE/s at 256 nodes (Scale 33)

ETI’s SWARM achieves the highest performance per core at the single-chip level.
G-500: SWARM vs. MPI (Scalability)

SWARM/MPI on Endeavor

Edges/Second vs. #Nodes
G-500: SWARM vs MPI (portability)

- More than 100% improvement over MPI for large scale problems
Outline

• Introduction
• Fine-Grain Execution Models – A History Perspective
• Cyclops-64 Execution Model (TNT) and Bare-Metal OS Approach
• From TNT to Codelet: A Path to The UHPC/Runnemedede Execution Model
• SWARM - Preliminary Results (e.g. Graphs-500, etc.)
• Future Directions – Beyond Runnemedede
Monty’ Comments

• Story of one-of-a-kind project
• The tradeoffs/issues of time-to-proof-concept-prototype (2 year is not enough)
Acknowledgements

• Our Sponsors
• Intel UHPC Team
• CAPSL UHPC Team (Juergen, Kelly, Daniel, Elkin, Stéphane, Robert, Josh S, Josh L, Aaron, etc.)
• ETI UHPC Team (Rishi, et. al.)
• Other Collaborators
• My Host