System Software for Big Data Computing

Cho-Li Wang
The University of Hong Kong
HKU High-Performance Computing Lab.

- Total # of cores: 3004 CPU + 5376 GPU cores
- RAM Size: 8.34 TB
- Disk storage: 130 TB
- Peak computing power: 27.05 TFlops

GPU-Cluster (Nvidia M2050, “Tianhe-1a”): 7.62 Tflops

31.45 TFlops (X12 in 3.5 years)
Big Data: The "3Vs" Model

- **High Volume** (amount of data)
- **High Velocity** (speed of data in and out)
- **High Variety** (range of data types and sources)

**2010:** 800,000 petabytes (would fill a stack of DVDs reaching from the earth to the moon and back)

**By 2020,** that pile of DVDs would stretch half way to Mars.
Our Research

• Heterogeneous Manycore Computing (CPUs+ GUPs)
• Big Data Computing on Future Manycore Chips
• Multi-granularity Computation Migration
(1) Heterogeneous Manycore Computing (CPUs + GUPs)

JAPONICA: Java with Auto-Parallelization ON Graphics Coprocessing Architecture
Heterogeneous Manycore Architecture

CPUs

GPU
## New GPU & Coprocessors

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Model</th>
<th>Launch Date</th>
<th>Fab. (nm)</th>
<th>#Accelerator Cores (Max.)</th>
<th>GPU Clock (MHz)</th>
<th>TDP (watts)</th>
<th>Memory</th>
<th>Bandwidth (GB/s)</th>
<th>Programming Model</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel</td>
<td>Sandy Bridge</td>
<td>2011Q1</td>
<td>32</td>
<td>12 HD graphics <strong>3000 EUs</strong> (8 threads/EU)</td>
<td>850 – 1350</td>
<td>95</td>
<td>L3: 8MB Sys mem (DDR3)</td>
<td>21</td>
<td>OpenCL</td>
<td>Bandwidth is system DDR3 memory bandwidth</td>
</tr>
<tr>
<td>Intel</td>
<td>Ivy Bridge</td>
<td>2012Q2</td>
<td>22</td>
<td>16 HD graphics <strong>4000 EUs</strong> (8 threads/EU)</td>
<td>650 – 1150</td>
<td>77</td>
<td>L3: 8MB Sys mem (DDR3)</td>
<td>25.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMD</td>
<td>Xeon Phi</td>
<td>2012H2</td>
<td>22</td>
<td><strong>60 x86 cores</strong> (with a 512-bit vector unit)</td>
<td>600-1100</td>
<td>300</td>
<td>8GB GDDR5</td>
<td><strong>320</strong></td>
<td>OpenMP#, OpenCL*, OpenACC%</td>
<td>Less sensitive to branch divergent workloads</td>
</tr>
<tr>
<td>AMD</td>
<td>Brazos 2.0</td>
<td>2012Q2</td>
<td>40</td>
<td>80 Evergreen shader cores</td>
<td>488-680</td>
<td>18</td>
<td>L2: 1MB Sys mem (DDR3)</td>
<td>21</td>
<td>OpenCL, C++AMP</td>
<td></td>
</tr>
<tr>
<td>AMD</td>
<td>Trinity</td>
<td>2012Q2</td>
<td>32</td>
<td>128-384 Northern Islands cores</td>
<td>723-800</td>
<td>17-100</td>
<td>L2: 4MB Sys mem (DDR3)</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nvidia</td>
<td>Fermi</td>
<td>2010Q1</td>
<td>40</td>
<td>512 Cuda cores (16 SMs)</td>
<td>1300</td>
<td>238</td>
<td>L1: 48KB L2: 768KB 6GB</td>
<td><strong>148</strong></td>
<td>CUDA, OpenCL, OpenACC</td>
<td></td>
</tr>
<tr>
<td>Nvidia</td>
<td>Kepler (GK110)</td>
<td>2012Q4</td>
<td>28</td>
<td><strong>2880 Cuda cores</strong></td>
<td>836/876</td>
<td>300</td>
<td>6GB GDDR5</td>
<td><strong>288.5</strong></td>
<td>CUDA, OpenCL, OpenACC</td>
<td>3X Perf/Watt, Dynamic Parallelism, HyperQ</td>
</tr>
</tbody>
</table>
#1 in Top500 (11/2012): Titan @ Oak Ridge National Lab.

- **18,688** AMD Opteron 6274 16-core CPUs (32GB DDR3).
- **18,688** Nvidia Tesla K20X GPUs
- Total RAM size: over 710 TB
- Total Storage: 10 PB.
- **Peak Performance: 27 Petaflop/s**
  - GPU: CPU = 1.311 TF/s: 0.141 TF/s = **9.3 : 1**
- **Linpack: 17.59 Petaflop/s**
- **Power Consumption: 8.2 MW**

Titan compute board: 4 AMD Opteron + 4 NVIDIA Tesla K20X GPUs

NVIDIA Tesla K20X (Kepler GK110) GPU: **2688** CUDA cores
Design Challenge: GPU Can’t Handle Dynamic Loops

GPU = SIMD/Vector
Data Dependency Issues (RAW, WAW)

Static loops
for(i=0; i<N; i++)
{
    C[i] = A[i] + B[i];
}

Dynamic loops
for(i=0; i<N; i++)
{
}

Non-deterministic data dependencies inhibit exploitation of inherent parallelism; only DO-ALL loops or embarrassingly parallel workload gets admitted to GPUs.
Dynamic loops are common in scientific and engineering applications

Lots of dynamic DO-ALL loops!

Source: Z. Shen, Z. Li, and P. Yew, "An Empirical Study on Array Subscripts and Data Dependencies"
GPU-TLS: Thread-level Speculation on GPU

• Incremental parallelization
  o sliding window style execution.
• Efficient dependency checking schemes
• Deferred update
  o Speculative updates are stored in the write buffer of each thread until the commit time.
• 3 phases of execution

Phase I
• Speculative execution
Phase II
• Dependency checking
Phase III
• Commit

GPU: lock-step execution in the same warp (32 threads per warp).
JAPONICA: Profile-Guided Work Dispatching

Dynamic Profiling → Scheduler

Dependency density

High → Parallel
8 high-speed x86 cores
Multi-core CPU

Medium → Highly parallel
64 x86 cores

Low/None → Massively parallel
2880 cores

Inter-iteration dependence:
-- Read-After-Write (RAW)
-- Write-After-Read (WAR)
-- Write-After-Write (WAW)

Dynamic Profiling

Multi-core CPU

8 high-speed x86 cores

64 x86 cores

2880 cores

Many-core coprocessors
Sequential Java Code with user annotation

\[ \text{JavaR} \]

- Code Translation
- Static Dep. Analysis

DO-ALL Parallelizer

- CPU-Multi threads
- GPU-Many threads

CUDA kernels & CPU Multi-threads

Task Scheduler: CPU-GPU Co-Scheduling

- Task Sharing
  - High DD: CPU single core
  - Low DD: CPU+GPU-TLS
  - o: CPU multithreads + GPU

- Task Stealing
  - CPU queue: low, high, o
  - GPU queue: low, o

Profiler (on GPU)

- Dependency Density Analysis
  - Intra-warp Dep. Check
  - Inter-warp Dep. Check

Speculator

- GPU-TLS
- Privatization

Profiling Results

Program Dependence Graph (PDG)

RAW

WAW/WAR

Profiling Results

one loop

Assign the tasks among CPU & GPU according to their dependency density (DD)
(2) Crocodiles: Cloud Runtime with Object Coherence On Dynamic tILES”
“General Purpose” Manycore

<table>
<thead>
<tr>
<th>Micro-architecture</th>
<th># of cores</th>
<th>On-Chip Network (Link Bandwidth)</th>
<th>H/W Coherence</th>
<th>L1$/core</th>
<th>L2$/core</th>
<th>L3$</th>
<th>DDR Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teraflops Research Chip</td>
<td>80 (4.0 GHz)</td>
<td>2D Mesh (256Gb/s)</td>
<td>No</td>
<td>5KB</td>
<td>256KB</td>
<td>NA</td>
<td>3D stacked memory</td>
</tr>
<tr>
<td>MIT’s ATAC (2008)</td>
<td>1000 (simulation)</td>
<td>2D (optical) Mesh (32Gb/s)</td>
<td>Yes</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Single-Chip Cloud (2009)</td>
<td>48 (1.0 GHz)</td>
<td>2D Mesh (512Gb/s)</td>
<td>No</td>
<td>32KB</td>
<td>256KB + 8KB MPB</td>
<td>Nil</td>
<td>4</td>
</tr>
<tr>
<td>Tilera Tile-GX (2009)</td>
<td>100 (1.5 GHz)</td>
<td>2D Mesh (320Gb/s)</td>
<td>Yes</td>
<td>64KB</td>
<td>256KB</td>
<td>26MB (shared)</td>
<td>4</td>
</tr>
<tr>
<td>Godson-T (FPGA, 2011)</td>
<td>64 (1.0 GHz)</td>
<td>2D Mesh</td>
<td>Yes</td>
<td>32KB</td>
<td>128KB x 16 shared</td>
<td>Nil</td>
<td>4</td>
</tr>
</tbody>
</table>

**Tile-based architecture:** Cores are connected through a 2D network-on-a-chip
Crocodiles: Cloud Runtime with Object Coherence On Dynamic tiles for future 1000-core tiled processors
• **Dynamic Zoning**
  - Multi-tenant Cloud Architecture → Partition varies over time, mimic “Data center on a Chip”.
  - Performance isolation
  - On-demand scaling.
  - Power efficiency (high flops/watt).
Design Challenge: “Off-chip Memory Wall” Problem

- DRAM performance (latency) improved slowly over the past 40 years.

(a) Gap of DRAM Density & Speed
(b) DRAM Latency Not Improved

Memory density has doubled nearly every two years, while performance has improved slowly (e.g. still 100+ of core clock cycles per memory access)
Lock Contention in Multicore System

- Physical memory allocation performance sorted by function. As more cores are added more processing time is spent contending for locks.
Exim on Linux

**Kernel CPU time (milliseconds/message)**

**Throughput (messages/second)**

**Core Count**

[Graph showing throughput and kernel CPU time versus core count with a notable 'collapse' area.]
Challenges and Potential Solutions

• **Cache-aware design**
  - Data Locality/Working Set getting critical!
  - Compiler or runtime techniques to improve data reuse

• **Stop multitasking**
  - Context switching breaks data locality
  - Time Sharing → Space Sharing

马其顿方阵众核操作系统：Next-generation Operating System for 1000-core processor
Thanks!

For more information:

C.L. Wang’s webpage:
http://www.cs.hku.hk/~clwang/
We have a few PhD (or 4-year 直博生) positions open for self-motivated and academically strong students this year. If you are interested in one of the projects, please contact me at clwang@cs.hku.hk. Interview will be arranged for qualified students.

1. **Crocodiles: Scalable Cloud-on-Chip Runtime Support with Software Coherence for Future 1000-Core Tiled Architectures**, HKU 716712E, 9/2012-8/2015, supported by HK RGC.

   Moving up to a parallelism with 1,000 cores requires a fairly radical rethinking of how to design system software. With a growing number of cores, providing hardware-level cache coherence gets increasingly complicated and costly, leading researchers to promote abandoning it if future many-core architectures are to stay inherently scalable. That means software now has to take on the role in ensuring data coherence among cores, yet exposing the low-level core-to-core message passing interfaces to programmers for managing coherence hampers programmability considerably. In this research, we address the above issues and propose novel methodologies to build a scalable CoC runtime platform, dubbed *Crocodiles* (*Cloud Runtime with Object Coherence On Dynamic tiles*), for future 1000-core tiled processors. *Crocodiles* involves the development of two important software subsystems: (1) Cache coherency protocol (2) *DVFS*-based power management.

   - **2 Ph.D students (highest priority):** strong background in OS kernel, full knowledge in memory subsystem (cache/DRAM, paging), cache coherent protocols.
   - **1-2 RAs:** Require strong background in software distributed shared memory systems (e.g., TreadMarks, JiaJa, JUMP), programming experiences in multicore power management systems. Starting date: **ASAP**.

2. **Japonica: Transparent Runtime and Memory Coherence Support for GPU Based Heterogeneous Many-Core Architecture**, 11/2011-10/2013, supported by HK RGC.

   In this project, we propose a new runtime platform, called *Japonica* (Java with Auto-Parallelization ON Graphics Co-processing Architecture), which enables a multithreaded Java program to scale transparently on a GPU-based heterogeneous system. With the transparent runtime support, application developers can utilize both CPU and GPU resources seamlessly with an idiomatic Java programming model. Japonica possesses several unique features: (1) automatic translation from Java bytecode to OpenCL, (2) auto-parallelization of loops with non-deterministic data dependencies, (3) dynamic load scheduling and rebalancing via task migration between CPU and GPU, (4) virtual shared memory between host and device, and (5) speculative coherency protocol for threads running on both CPU and GPU cores. The proposed work explores GPU-friendly ways to support a partial Java heap and STM-based synchronization of shared objects and arrays mirrored in GPU.

   - **1 Ph.D student**: interested in Java Virtual Machine, compiler (e.g., loop parallelization), software transactional memory. Experiences in GPU programming (CUDA or OpenCL) is required.
   - **RAs**: We are now building *Japonica* on a multicore GPU Cluster. We need 1 or 2 RAs who are good in OpenCL or CUDA programming.


3. **Self-Organizing Desktop Cloud (SODC)**

   We are currently building a P2P Personal Cloud system which heavily adopts the Xen virtual machines. The research focus will be on (1) **VM I/O performance isolation**, (2) **live VM migration over WAN**. Read [WAVMNet](http://www.cs.hku.hk/~clwang/projects/WAVMN.html) webpage for more details.

   - **1 PhD student**: Look for a strong candidate with good knowledge in virtual machines internal design (e.g., KVM, Xen), solid background in Linux kernel, and good knowledge in NAT/firewall tunneling solutions.

4. **OS-1K: New Operating System for Manycore Systems (part of Crocodiles project)**

   Traditional operating systems are based on the sequential execution model developed in the 1950s. Such operating systems cannot address new many-core parallel hardware architecture without major redevelopment. For instance, how can you harness the power a next-generation manycore processor with >1,000 cores? We will investigate various perspectives on the future OS design towards the goal.
Multi-granularity Computation Migration

Granularity

Coarse

Fine

System scale
(Size of state)

Small

Large

Granularity

Frame level

Thread level

Process level

VM level

Migration Technique (System)

Stack-on-demand (SODEE)

Thread migration (JESSICA2)

Process migration (G-JavaMPI)

Live VM migration (Xen)

Wide-area live VM migration (WAVNet)

Target System Type (Area)

Cloud, cloudlet, mobile network (WAN/LAN)

Cluster (LAN)

Grid (WAN/LAN)

Cluster (LAN)

Cloud, p2p/desktop cloud (WAN)

WAVNet Desktop Cloud

G-JavaMPI

JESSICA2

SOD
WAVNet: Live VM Migration over WAN

- A P2P Cloud with Live VM Migration over WAN
  - “Virtualized LAN” over the Internet”
- High penetration via NAT hole punching
  - Establish direct host-to-host connection
  - Free from proxies, able to traverse most NATs

Key Members

WAVNet: Experiments at Pacific Rim Areas

<table>
<thead>
<tr>
<th>Sites</th>
<th>RTT (ms)</th>
<th>WAVNet bw (Mbps)</th>
<th>Time taken (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OffCam-HKU</td>
<td>4.4</td>
<td>86.39</td>
<td>16</td>
</tr>
<tr>
<td>Sinica-HKU</td>
<td>24.8</td>
<td>42.93</td>
<td>92.5</td>
</tr>
<tr>
<td>AIST-HKU</td>
<td>75.8</td>
<td>55.1</td>
<td>107.5</td>
</tr>
<tr>
<td>SIAT-HKU</td>
<td>74.2</td>
<td>18.6</td>
<td>130</td>
</tr>
<tr>
<td>SDSC-HKU</td>
<td>217.2</td>
<td>27.17</td>
<td>310.5</td>
</tr>
</tbody>
</table>
JESSICA2: Distributed Java Virtual Machine

A Multithreaded Java Program

Thread Migration

Portable Java Frame

JIT Compiler Mode

Java Enabled Single System Image Computing Architecture
History and Roadmap of JESSICA Project

• **JESSICA V1.0 (1996-1999)**
  – Execution mode: **Interpreter Mode**
  – JVM kernel modification (Kaffe JVM)
  – Global heap: built on top of TreadMarks (Lazy Release Consistency + homeless)

• **JESSICA V2.0 (2000-2006)**
  – Execution mode: **JIT-Compiler Mode**
  – JVM kernel modification
  – Lazy release consistency + migrating-home protocol

• **JESSICA V3.0 (2008~2010)**
  – **Built above JVM (via JVMTI)**
  – Support Large Object Space

• **JESSICA v.4 (2010~)**
  – **Japonica**: Automatic loop parallelization and speculative execution on GPU and multicore CPU
  – **TrC-DC**: a software transactional memory system on cluster with distributed clocks (not discussed)

J1 and J2 received a total of **1107** source code downloads
Stack-on-Demand (SOD)
Elastic Execution Model via SOD

(a) “Remote Method Call”
(b) Mimic thread migration
(c) “Task Roaming”: like a mobile agent roaming over the network or workflow

With such flexible or **composable** execution paths, SOD enables agile and elastic exploitation of distributed resources (storage), a Big Data Solution!

Lightweight, Portable, Adaptable