System Software Challenges for Big Data Computing

Cho-Li Wang
The University of Hong Kong
Systems Research Group @HKU

Our Motto: solving **REAL** problems with the use of **REAL** computing resources
“Self-Made” Gideon 300 cluster in 2002

- 300 Pentium4 PCs @355 Gflops;
- Ranked #170 in TOP500 (11/2002), #3 in China.
- The highest ranking in the TOP500 list among all machines from Hong Kong academic institutions in history.
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

CS Gideon-II & CC MDRP Clusters

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

31.45TFlops (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.
McKinsey Global Institute (MGI):
- Using big data, retailers could increase its operating margin by more than 60%.
- The U.S. could reduce its healthcare expenditure by 8%
- Government administrators in Europe could save more than €100 billion ($143 billion).
2012 CIO Agenda Findings

Success is contingent on anticipating the coming changes

2,335 CIOs from 37 industries across 45 countries

<table>
<thead>
<tr>
<th>CIO technologies</th>
<th>Ranking of technologies CIOs selected as one of their top 3 priorities in 2012</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>2012</strong></td>
<td><strong>2011</strong></td>
</tr>
<tr>
<td>Analytics and business intelligence</td>
<td>1</td>
</tr>
<tr>
<td>Mobile technologies</td>
<td>2</td>
</tr>
<tr>
<td>Cloud computing (SaaS, IaaS, PaaS)</td>
<td>3</td>
</tr>
<tr>
<td>Collaboration technologies (workflow)</td>
<td>4</td>
</tr>
<tr>
<td>Virtualization</td>
<td>5</td>
</tr>
<tr>
<td>Legacy modernization</td>
<td>6</td>
</tr>
<tr>
<td>IT management</td>
<td>7</td>
</tr>
<tr>
<td>Customer relationship management</td>
<td>8</td>
</tr>
<tr>
<td>ERP applications</td>
<td>9</td>
</tr>
<tr>
<td>Security</td>
<td>10</td>
</tr>
<tr>
<td>Social media/Web 2.0</td>
<td>11</td>
</tr>
</tbody>
</table>

*Not an option that year

(Big Data)
Outline

• Part I: Multi-granularity Computation Migration
• Part II: Heterogeneous Manycore Computing (CPUs+ GUPs)
• Part III: Big Data Computing on Future 1000-core Chips
• Part IV: From Data to Intelligence -- Context Reasoning
Part I

Multi-granularity Computation Migration
Multi-granularity Computation Migration

<table>
<thead>
<tr>
<th>Granularity</th>
<th>Migration Technique (System)</th>
<th>Target System Type (Area)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame level</td>
<td>Stack-on-demand (SODEE)</td>
<td>Cloud, cloudlet, mobile network (WAN/LAN)</td>
</tr>
<tr>
<td>Thread level</td>
<td>Thread migration (JESSICA2)</td>
<td>Cluster (LAN)</td>
</tr>
<tr>
<td>Process level</td>
<td>Process migration (G-JavaMPI)</td>
<td>Grid (WAN/LAN)</td>
</tr>
<tr>
<td>VM level</td>
<td>Live VM migration (Xen)</td>
<td>Cluster (LAN)</td>
</tr>
<tr>
<td></td>
<td>Wide-area live VM migration (WAVNet)</td>
<td>Cloud, p2p/desktop cloud (WAN)</td>
</tr>
</tbody>
</table>

Granularity: Coarse
Fine

System scale: Small → Large
(Size of state)
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: Live VM Migration over WAN

• 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></td>
<td></td>
<td></td>
<td>128M</td>
</tr>
<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>
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 parallization 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
### Comparison of Migration Overhead

Migration overhead (MO) = execution time w/ migration – execution time w/o migration

<table>
<thead>
<tr>
<th>App</th>
<th>Exec. time (sec) w/ mig</th>
<th>Exec. time (sec) w/o mig</th>
<th>MO (ms) w/ mig</th>
<th>MO (ms) w/o mig</th>
<th>Exec. time (sec) w/ mig</th>
<th>Exec. time (sec) w/o mig</th>
<th>MO (ms) w/ mig</th>
<th>MO (ms) w/o mig</th>
<th>Exec. time (sec) w/ mig</th>
<th>Exec. time (sec) w/o mig</th>
<th>MO (ms) w/ mig</th>
<th>MO (ms) w/o mig</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fib</td>
<td>12.77</td>
<td>12.69</td>
<td>83</td>
<td></td>
<td>47.31</td>
<td>47.21</td>
<td>96</td>
<td></td>
<td>16.45</td>
<td>12.68</td>
<td>3770</td>
<td></td>
</tr>
<tr>
<td>NQ</td>
<td>7.72</td>
<td>7.67</td>
<td>49</td>
<td></td>
<td>37.49</td>
<td>37.30</td>
<td>193</td>
<td></td>
<td>7.93</td>
<td>7.63</td>
<td>299</td>
<td></td>
</tr>
<tr>
<td>TSP</td>
<td>3.59</td>
<td>3.58</td>
<td>13</td>
<td></td>
<td>19.54</td>
<td>19.44</td>
<td>96</td>
<td></td>
<td>3.67</td>
<td>3.59</td>
<td>84</td>
<td></td>
</tr>
<tr>
<td>FFT</td>
<td>10.79</td>
<td>10.60</td>
<td>194</td>
<td></td>
<td>253.63</td>
<td>250.19</td>
<td>3436</td>
<td></td>
<td>15.13</td>
<td>10.75</td>
<td>4379</td>
<td></td>
</tr>
</tbody>
</table>

**SOD has the smallest migration overhead:** ranges from **13ms** to **194ms** under Gigabit Ethernet


Part II

Heterogeneous Manycore Computing (CPUs+ GUPs)

**JAPONICA**: Java with Auto-Parallelization **ON** Graphics **Coprocessing** Architecture
Heterogeneous Manycore Architecture
## A Variety of 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 3000 EUs (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></td>
<td>Ivy Bridge</td>
<td>2012Q2</td>
<td>22</td>
<td>16 HD graphics 4000 EUs (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></td>
<td>Xeon Phi</td>
<td>2012H2</td>
<td>22</td>
<td>57 x86 cores (with a 512-bit vector unit)</td>
<td>600-1100</td>
<td>300</td>
<td>8GB GDDR5</td>
<td>300</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></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>APU</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>148</td>
<td>CUDA, OpenCL, OpenACC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Kepler</td>
<td>2012Q2</td>
<td>28</td>
<td>1536 Cuda cores</td>
<td>1000</td>
<td>300</td>
<td>8GB GDDR5</td>
<td>320</td>
<td></td>
<td>3X Perf/Watt, Dynamic Parallelism, HyperQ</td>
</tr>
</tbody>
</table>
Intel Many Integrated Core Architecture (MIC)

**Larrabee (2006-2010)**
- Single-chip Cloud Computer (2009-) 48 cores
- Teraflops Research Chip (2007) 80 cores, 3.16GHz, 1.01 Tflops, 62W

**Knights Ferry (2010)**
- 32 cores, 1.2 GHz, 750 GFLOPS, 2 GB GDDR5, ~300 W

**Knights Corner (2012)**
- 22nm 50+ cores,

The MIC chip has a superscalar x64 core (without the out-of-order execution of Xeons) and a 512-bit vector math unit that can do 16 floating point operations per clock with single precision math.

'Standard's Corner' chips (branded as 'Xeon Phi')- 6/2012 -- 64 x86 cores (256 threads) + a 512-bit vector unit @2GHz, 1 Teraflops
Design Challenge (1)

1. Copy input data from CPU memory to GPU memory (e.g., cudaMemcpy())
2. Load GPU program and execute, caching data on chip for performance
3. Copy results from GPU memory to CPU memory

PCI Bus

Bottleneck: PCIe gen2 peak bandwidth = 8 GB/s

Fermi GPU load/store DRAM peak bandwidth = 148 GB/s (MIC=300 GB, Kepler=320 GB)
Does PCIe 3.0 help?

Informal testing results: No appreciable difference in performance between PCIe 3 x16 (16GB/sec) and PCIe 2 (8GB/sec)

Require much higher “flops per byte” – i.e., applications with “High Arithmetic Intensity” (HAI)
Solution? : CPU-GPU mashups

AMD "Trinity".

AMD’s new **Accelerated Processing Units** combine general-purpose x86 CPU cores with programmable vector processing engines on a single silicon die.

**Ivy Bridge** GPU incorporates a high bandwidth **L3 cache** that is shared by the entire shader array.
Design Challenge (2): GPU Can’t Handle Dynamic Loops

**GPU = SIMD/Vector**

Data Dependency Issues (RAW, WAW)

Static loops

```c
def(i=0; i<N; i++)
{
    C[i] = A[i] + B[i];
}
```

Dynamic loops

```c
for(i=0; i<N; i++)
{
    A[w[i]] = 3 * A[r[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.

Source: Z. Shen, Z. Li, and P. Yew, "An Empirical Study on Array Subscripts and Data Dependencies"
Thread Level Speculation (TLS)

- Execute hard-to-analyze codes speculatively (or optimistically) in parallel.
  - Assume no dependences and execute in parallel
  - Track memory accesses and detect violations
  - Squash and restart offending threads
Thread Level Speculation (TLS)

- Execute hard-to-analyze codes speculatively (or optimistically) in parallel.
  - Assume no dependences and execute in parallel
  - Track memory accesses and detect violations
  - Squash and restart offending threads

```plaintext
for(i=0;i<n;i++)
    ...=a[b[i]]...
    ...
    a[c[i]]=...
    a[5]=...
    ...=a[5]...
    ...
    a[6]=...
    ...
    a[6]=...
```

```
i=k  i=k+1  i=k+2  i=k+3
...=a[4]...  ...=a[1]...  ...=a[8]...  ...=a[5]...
    ...
    a[2]=...
    a[9]=...
    ...
    a[6]=...
    ...
```
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).

① intra-thread RAW ✓
② valid inter-thread RAW in GPU ✓
③ true inter-thread RAW ✗

Thread 0
- Write A[0]
- Write A[1]
- Write A[2]
- Read A[0]
- Read A[3]
- Write A[3]

Thread 30
- Write A[90]
- Write A[91]
- Write A[92]
- Read A[89]
- Read A[90]
- Write A[2]

Thread 31
- Write A[90]
- Write A[1]
- Write A[92]
- Read A[90]
- Read A[2]
- Write A[3]
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)

Multi-core CPU → Many-core coprocessors
**JAPONICA : System Architecture**

Sequential Java Code with user annotation

**JavaR**
- Code Translation
- Static Dep. Analysis

**DO-ALL Parallelizer**
- CPU-Multi threads
- GPU-Many threads

**Profiler (on GPU)**
- Dependency Density Analysis
  - Intra-warp Dep. Check
  - Inter-warp Dep. Check
- Profiling Results
- Speculator
  - GPU-TLS
  - Privatization
- RAW
- WAW/WAR

**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

Assign the tasks among CPU & GPU according to their *dependency density (DD)*

**Communication**
- CPU
- GPU
GPU-TLS: Performance Evaluation

(a) Speedups for DOALL loops
(b) Speedups for WAW loops
(c) Speedups for WAR loops
(d) Speedups for RAW loops
**JAPONICA Evaluation: Bi-Conjugate Gradient (BICG)**

<table>
<thead>
<tr>
<th>Number of iterations</th>
<th>CPU</th>
<th>CPU_16</th>
<th>GPU</th>
<th>CPU+GPU</th>
<th>Workload of CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>2048</td>
<td>44.792</td>
<td>5.091</td>
<td>5.521</td>
<td>2.891</td>
<td>50%</td>
</tr>
<tr>
<td>3072</td>
<td>100.016</td>
<td>12.269</td>
<td>14.391</td>
<td>10.482</td>
<td>50%</td>
</tr>
<tr>
<td>4096</td>
<td>179.699</td>
<td>19.791</td>
<td>18.856</td>
<td>14.004</td>
<td>50%</td>
</tr>
<tr>
<td>7168</td>
<td>544.005</td>
<td>50.841</td>
<td>52.668</td>
<td>28.332</td>
<td>50%</td>
</tr>
<tr>
<td>8192</td>
<td>718.596</td>
<td>65.32</td>
<td>60.205</td>
<td>34.691</td>
<td>50%</td>
</tr>
<tr>
<td>10240</td>
<td>1109.6</td>
<td>100.871</td>
<td>135.728</td>
<td>65.044</td>
<td>62.5%</td>
</tr>
</tbody>
</table>

**from the Polybench**
General Observations and Prediction

- **Lowering clock rate** but many more cores.
  - Kepler 1 Ghz (3072) vs Fermi 1.3 Ghz (512)
- **More power efficient** (increasing perf/watt)
- **Increasing bandwidth** (> 300 GB/s, e.g., Kepler)
  - getting readier for data intensive workloads.
- **More dynamic workflow:**
  - **Kepler’s Dynamic Parallelism**: GPU kernel can spawn new work onto the GPU
- **Intel MIC**, using x86 cores, is stealing the limelight.
  - We foresee it will be a norm in the coprocessor world.
  - Deliver similar flops (1 Tflops) but easier programming
Part III

Big Data Future 1000-core
“General Purpose” Maycore Chips
### “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><strong>No</strong></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><strong>No</strong></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
Tiled Manycore Architectures

- The cores of the SCC are grouped into multiple domains in terms of frequency, voltage and memory access.

Multiple cores per tile, connected by an on-die 2D mesh network (network-on-chip).
Design Challenge (1):
“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)
Design Challenge (2): “Coherency Wall” Problem

Overhead of enforcing cache coherency across 1,000 cores at hardware level will put a hard limit on scalability

1. **Performance overhead**: Coherence uses 20% more traffic per miss than a system with caches but not coherence
2. **Die space overhead**: cache directory, read/write log increase
3. **Not always needed**: Only around 10% of the application memory references actually require cache coherence tracking
4. **Verification complexity and extensibility**: require dealing with subtle races and many transient states

Intel’s SCC and Teraflops Research Chip decided to give up coherent caches.
Laser-Powered Chip in 2017??

HP Corona: 10-Teraflop Manycore Chip (expected 2017)

- 256 cores, each supporting up to four threads
- **Optical interconnect**: a **20 TB/sec** DWDM crossbar
- **Optically connected memory (OCM)**: **10 TB/sec**
  - 80 GB/sec: 8-core Intel E5-2600 Xeons
  - 64 GB/sec: SPARC64 VIIIfx CPU of K computer
  - 177 GB/sec: NVIDIA M2090,
- **Energy efficiency**: 6.4 watts @ 10 GB/sec of data to DRAM, which is 25 x less than electrical interconnect (160 watts)
- **MOESI directory cache coherency protocol**
- Aim at big data applications
- **Other projects**: Intel’s Runnemedede, MIT’s Angstrom, NVIDIA’s Echelon, and Sandia’s X-calibur.
Design Challenge (3): “Power Wall” Problem

• Computation costs much less energy than moving data to and from the computation units
• As the energy cost of computation is reduced by voltage scaling, the cost of data movement starts to dominate.

If only 10% of the operands move over the network, 10 hops in average, at 0.06pJ/bit the network would consume 35 watts of power, > 50% of the power budget of the processor.

• Bill Dally, Chief Scientist of nVIDIA
  o 1 pJ for an integer operation
  o 20 pJ for a floating-point operation
  o 26 pJ to move an operand over 1mm of wire to local memory
  o 1 nJ to read an operand from on-chip memory located at the far end of a chip
  o 16 nJ to read an operand from off-chip DRAM

You cannot break the laws of physics - and 7nm is the limit

On-die network energy consumption per bit

picojoule (pJ) = 10^{-12} J
nanojoule (nJ) = 10^{-9} J
Design Challenge (4): OS Scalability

Fig. 4. Normalized transaction throughput scalability of Sysbench-OLTP and TPCC-UVa with the number of cores.

Y. Cui, et al, Scaling OLTP Applications on Commodity Multi-Core Platforms, ISPASS10
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.
Linux Atomic Operations

• x86 LOCK prefix makes many read-modify-write instructions atomic.
• Most general instruction is cmpxchg, used to implement locks

```c
29 static __inline__ void atomic_add(int i, atomic_t *v)
30 {
    __asm__ __volatile__ ("LOCK "addl $1,$0"
31 :="m" (v->counter)
32 :="ir" (i), "m" (v->counter));
34
35 }
```

```c
37 static __inline__ void atomic_sub(int i, atomic_t *v)
38 {
    __asm__ __volatile__ ("LOCK "subl $1,$0"
39 :="m" (v->counter)
40 :="ir" (i), "m" (v->counter));
42
43 }
```

x86: LOCK prefix “locks” the memory bus for the destination memory address to allow the processor has exclusive use of any shared data.
How often is ‘cmpxchg’ used in Linux kernel?

$ cat vmlinux.asm | grep cmpxchg

c01046de:    f0 0f b1 15 3c 99 30    lock cmpxchg %edx,0xc030993c
c0105591:    f0 0f b1 15 3c 99 30    lock cmpxchg %edx,0xc030993c
c01055d9:    f0 0f b1 15 3c 99 30    lock cmpxchg %edx,0xc030993c
c010b895:    f0 0f b1 11    lock cmpxchg %edx,0xc030993c
c010b949:    f0 0f b1 0b    lock cmpxchg %edx,0xc030993c
c0129a9f:    f0 0f b1 0b    lock cmpxchg %edx,0xc030993c
c0129acf:    f0 0f b1 0b    lock cmpxchg %edx,0xc030993c
c012d377:    f0 0f b1 0e    lock cmpxchg %edx,0xc030993c
c012d41a:    f0 0f b1 0e    lock cmpxchg %edx,0xc030993c
c012d968:    f0 0f b1 16    lock cmpxchg %edx,0xc030993c
c012e568:    f0 0f b1 2e    lock cmpxchg %edx,0xc030993c
c012e5a7:    f0 0f b1 2e    lock cmpxchg %edx,0xc030993c
c012e58a:    f0 0f b1 2e    lock cmpxchg %edx,0xc030993c
c012e83f:    f0 0f b1 13    lock cmpxchg %edx,0xc030993c
c012e931:    f0 0f b1 0a    lock cmpxchg %edx,0xc030993c
c012ea94:    f0 0f b1 11    lock cmpxchg %edx,0xc030993c
c012ecf4:    f0 0f b1 13    lock cmpxchg %edx,0xc030993c
c012f08e:    f0 0f b1 4b 18    lock cmpxchg %edx,0xc030993c
c012f163:    f0 0f b1 11    lock cmpxchg %edx,0xc030993c
c013cb60:    f0 0f b1 0e    lock cmpxchg %edx,0xc030993c
c0148b3c:    f0 0f b1 29    lock cmpxchg %edx,0xc030993c
c0150d0f:    f0 0f b1 3b    lock cmpxchg %edx,0xc030993c
c0150d87:    f0 0f b1 31    lock cmpxchg %edx,0xc030993c
c0199c5e:    f0 0f b1 0b    lock cmpxchg %edx,0xc030993c
c024b06f:    f0 0f b1 0b    lock cmpxchg %edx,0xc030993c
c024b2fe:    f0 0f b1 51 18    lock cmpxchg %edx,0xc030993c
c024b321:    f0 0f b1 51 18    lock cmpxchg %edx,0xc030993c
c024b34b:    f0 0f b1 4b 18    lock cmpxchg %edx,0xc030993c
c024b960:    f0 0f b1 53 18    lock cmpxchg %edx,0xc030993c

Referenced in 25 files total (2.6.31.13)!
Operating Systems for Many-core (1)

• MIT Factored Operation System (fOS): 2009
  o Target 1,000 core multicore chip
  o Space sharing replaces time sharing
• Berkeley Tessellation (2009)
  o "Cell“ replace processes for performance isolation and QoS guarantees
• Microsoft: Barrelfish
  o Multikernel design: Build OS as a distributed system over all cores. Message passing among cores.
• Berkeley ROS (2010)
  o Space and time partitioning
  o ‘many-core’ process (MCP) abstraction
MIT fos: a Factored Operating System

- Space sharing replaces time sharing to increase scalability
- Mimic distributed Internet services
- fos’s system servers communicate via message passing

“Internet on a Chip”
Operating Systems for Many-core (2)

- **Microsoft Helios (2009)**
  - Running on **heterogeneous hardware**, based on Singularity OS
  - Satellite kernels, remote message passing, affinity

- **K42 (Since 1996): IBM, U of Toronto**
  - Microkernel architecture, **object-oriented design**, research purposes

- **Corey (2008): MIT & Fudan & Microsoft Research Asia**
  - Exo-kernel, re-implementing OS data structures (file descriptor table, mm_struct) and user APIs

- **µKMC (2012-): U. of Tokyo**
  - **Light-weight micro kernels on Intel MIC**, starts from July 2012.
  - Accelerator abstraction layer (AAL), inter-kernel communication layer (IKCL)

- **Berkeley Akaros (2010-2013)**
  - **Asymmetric OS structure** to scale to thousands of cores.
  - Per-core private memory, syscalls are ”context switch free”
Crocodiles: Cloud Runtime with Object Coherence On Dynamic tiles for future 1000-core tiled processors”
Challenges and Potential Solutions (1)

• **Stop moving so much data around**
  - Data Locality/Working Set getting critical!
  - 3D stacked memory (TSV technology) helps!
  - Compiler or runtime techniques to improve data reuse and increase **arithmetic intensity** (next slide)
  - Cache-aware design (**temporal locality** becomes more critical)
  - Migrating “code & state” instead of data → Thread migration among cores (+ large 3D stacked memory!).

• **Stop multitasking**
  - Context switching breaks data locality
  - No Time Sharing → Space Sharing
Arithmetic Intensity

- **Arithmetic intensity is defined as the number of operations performed per word of memory transferred**
- It is important for Big Data applications to have high arithmetic intensity, otherwise the memory access latency will limit computational speedup
Challenges and Potential Solutions (2)

- Software-managed cache coherence
  - Leverage programmable on-chip memory (e.g., MPB on Intel SCC)
  - Scope consistency (ScC): minimizing on-chip network and off-chip DRAM traffic
  - Migrating-home ScC Protocol (MH-ScC) → improve data locality

With home migration, each phase took much less execution time.

Simulation results obtained in a 8-node cluster (SOR program)
Challenges and Potential Solutions (3)

• Scalability (up to 1000 cores?)
  - Adopt multikernel operating system (e.g., Barrelfish) to reduce contentions on shared structures in OS kernel
  - Shared memory → message passing
    - **Barrelfish**: “Compact message cheaper than many cache lines-- even on a cache-coherent machine.”

View state as replicated: Maintain state through replication rather than shared memory (improved locality)
Challenges and Potential Solutions (4)

• **Dynamic Zoning for Elasticity of Demand**
  - “Zoning” (Spatial Partition) → Performance isolation
  - “Dynamic Zoning”: on-demand scaling of resources (e.g., # of cores, DRAM,..) for each zone.
  - Partitioning varies over time, mimic multi-tenant Cloud Architecture → “Data center on a Chip”
  - Fit well with the domain-based power management (e.g., Intel SCC)
Conclusion

• **GHz game is over → Go for Manycore**
  o World has gone to manycore to continue Moore’s Law
  o “General-purpose” 100-core chip is available (e.g., Tilera TILE-Gx), 1000-core chip is expected soon (2017?)
  o Intel MIC to be used in China’s 100 petaflops machine?

• **PCIe bottleneck problem?**
  o CPU-GPU mashup (e.g., APU)

• **Big data computing on 1000-core chip is tough**
  o **Locality is critical** (compute is “free”, avoid moving data around)
  o Power efficiency is the key challenge (flops/watt)
  o Low AI problem: Data reuse techniques for high **flops/byte**
Conclusion (Cont’d)

• **Scalability issues in all layers:**
  - Hardware (NoC), OS, software cache coherency, programming model
• **“DON’T MOVE THE DATA!”**
  - Implication: *moving code & state instead*
  - Try “Multi-granularity Computation Migration”
• **Research in system software is hard.**
  - There are rarely clearly right or clearly wrong solutions. No “**one-size-fits-all**” solution.
  - Difficult to compare: No standard interfaces
  - **Pressures from academic publication volume or deliverables**
Questions?
Part IV
From Data to Intelligence -- *Context Reasoning*
**Context Reasoning**

- *Data is only valuable when you can gain insights from it to make decisions*

- **Context Reasoning:**
  - deducing new and relevant information to the use of application(s) and user(s) from the various sources of context-data.

- These tasks include: (1) *context data pre-processing*, (2) *sensor data fusion* and (3) *mapping lower level context into higher level context* (which is also known as *context inference*).
Context Reasoning: Significant Places Detection

From lower-level raw data to meaningful higher-level context

(a) Seven extracted places:
- a: King George V Memorial Park
- b: 7-Eleven convenience store
- c: Pizza-Box store
- d: Bus station
- e: Flora Ho Sports Centre
- f: Pokfulam Road Playground
- g: a restaurant
Ontology-based Context Modeling

- Ontologies provide a **vocabulary** of terms
  - Meaning (*semantics*) of such terms is **formally specified**
  - New terms can be formed by **combining existing ones**
- **Focus on semantics and reasoning**!
post-Hadoop era (取代GFS 和 MapReduce)

• **Google Caffeine (2010)**
  - 主要为Google网络搜索引擎提供支持 (2010)
  - 将索引放置在由Google的分布式数据库BigTable上
• **Google Pregel (SIGMOD 2010)**
  - Large-scale graph processing (图形数据库)
• **Google Dremel (VLDB 2010):**
  - interactive ad-hoc query
  - 可以在几秒的时间处理PB级的数据查询 (BigQuery)
• **Google Percolator:**
  - for incremental processing (Bigtable)
• **Apache Giraph (Open Source)**
  - HDFS + Zookeeper
ContextTorrent
semantically organize, search, and store various types of contexts and their semantic relationships using ontology-based semantic technologies

Ontology-based Context Modeler

RDF triple: \((subject, predicate, object)\)

\(<\text{person:”Dexter”, hasAge, “25”}>\),
\(<\text{person: “Tom”, hasLocation, (110.1, 20.3)}>\)
\(<\text{hk_weatherstation, hasPredictinon, “Sunny”}>\)


ODBMS (db4o, Perst): object-orientation analogous to ontological representation
Evaluation

**Storage Consumption**

- **SQLite**
- **db4o**
- **Perst**

**Continuous Query**

**Randomly Query**

**Breakdown**
BetterLife 2.0: Large-scale Social Intelligence Reasoning

Case-based Reasoning

Cloud

Case-based Reasoning Engine (jCOLIBRI2)

Social Network

Dexter H. Hu, Yinfeng Wang, Cho-Li Wang, "BetterLife 2.0: Large-scale Social Intelligence in Cloud Computing" (CloudCom 2010)
WAVNet: Live VM Migration over WAN

**Ping RTT** drops after migration
Freeze Time: 0.6s ~ 2.1s

ICMP RTT and HTTP throughput during VM live migration
*(x represents ICMP packet loss)*