DAS-5: Harnessing the Diversity of Complex e-Infrastructures

Henri Bal
Vrije Universiteit Amsterdam
High-Performance Distributed Computing

- Distributed systems continue to change
  - Clusters, grids, clouds, heterogeneous systems
- Distributed applications continue to change
- Distributed programming continues to be notoriously difficult
- I build infrastructures and study programming systems and applications hand-in-hand
My background

- Cluster computing
  - Zoo (1994), Orca
- Wide-area computing
  - DAS-1 (1997), Albatross
- Grid computing
  - DAS-2 (2002), Manta, Satin
- eScience & optical grids
  - DAS-3 (2006), Ibis
- Many-core accelerators
  - DAS-4 (2010), MCL, Glasswing
  - DAS-5 (2015), Cashmere
Agenda

   • Unique aspects, the 5 DAS generations, organization
   • Earlier results and impact  [IEEE Computer 2015]

II. Current research on DAS studying diversity
   • A methodology for programming many-core accelerators
   • Big data (MapReduce) applications on accelerators

   DAS-1  DAS-2  DAS-3  DAS-4  DAS-5
What is DAS?

- Distributed common infrastructure for Dutch Computer Science
  - Distributed: multiple (4-6) clusters at different locations
  - Common: single formal owner (ASCI), single design team
    - Users have access to entire system
  - Dedicated to CS experiments (like Grid’5000)
    - Interactive (distributed) experiments, low resource utilization
    - Able to modify/break the hardware and systems
  - Dutch: small scale
About SIZE

- Only ~200 nodes in total per DAS generation
- Less than 1.5 M€ total funding per generation
- Johan Cruyff:
  - "ieder nadeel heb zijn voordeel"
  - Every disadvantage has its advantage
Small is beautiful

- We have superior wide-area latencies
  - “The Netherlands is a 2×3 msec country” (Cees de Laat, Univ. of Amsterdam)

- Able to build each DAS generation from scratch
  - Coherent distributed system with clear vision

- Despite the small scale we achieved:
  - 3 CCGrid SCALE awards, numerous TRECVID awards
  - >100 completed PhD theses
DAS generations: visions

- DAS-1: Wide-area computing (1997)
  - Homogeneous hardware and software
- DAS-2: Grid computing (2002)
  - Globus middleware
  - Dedicated 10 Gb/s optical links between all sites
- DAS-4: Clouds, diversity, green IT (2010)
  - Hardware virtualization, accelerators, energy measurements
- DAS-5: Harnessing diversity, data-explosion (June 2015)
  - Wide variety of accelerators, larger memories and disks
ASCI (1995)

- Research schools (Dutch product from 1990s), aims:
  - Stimulate top research & collaboration
  - Provide Ph.D. education (courses)
- ASCI: Advanced School for **Computing** and **Imaging**
  - About 100 staff & 100 Ph.D. Students
  - 16 PhD level courses
  - Annual conference
Organization

- ASCI steering committee for overall design
  - Chaired by Andy Tanenbaum (DAS-1) and Henri Bal (DAS-2 – DAS-5)
  - Representatives from all sites: Dick Epema, Cees de Laat, Cees Snoek, Frank Seinstra, John Romein, Harry Wijshoff
- Small system administration group coordinated by VU (Kees Verstoep)
  - Simple homogeneous setup reduces admin overhead
Historical example (DAS-1)

- Change OS globally from BSDI Unix to Linux
- Under directorship of Andy Tanenbaum

---

**The New York Times**

Belgian King, Unable to Sign Abortion Law, Takes Day Off

By PAUL L. MONTGOMERY, Special to The New York Times
Published: April 06, 1990

The civilian Government temporarily suspended King Baudouin I from power today after he declared that he could not, in good conscience as a Roman Catholic, sign a new law permitting abortion.

After declaring the King "unable to govern," the Cabinet assumed the King's powers and promulgated the abortion law, which was published in the official gazette. And it called the Chamber of Deputies and the Senate back from Easter vacation for a special session on Thursday.

The lawmakers will be asked to vote on the proposition that the 59-year-old King is once again able to govern. Approval is expected.

Seen as Acceptable Way Out

There was uproar in some political groups that have traditionally been opposed to the monarchy. But television and press comment tonight generally supported the apparent solution to the King's problem as an acceptable outcome.

Belgium has 25 political parties divided among speakers of French and Flemish and four self-governing regions. Solutions that give each participant a taste of victory in complicated circumstances are called "a la Belge" - in the Belgian manner.

Belgium, a country of 10 million people, sometimes puts itself forward at international gatherings as a good model to follow when big powers and unyielding ideologies cannot find common ground.

Today's events were the first time since the creation of Belgium and its constitutional monarchy in 1830 that the special powers of the Government had been used in such a way.
Financing

- NWO ``middle-sized equipment” program
  - Max 1 M€, very tough competition, but scored 5-out-of-5
  - 25% matching by participating sites
    - Going Dutch for ¼ th
- Extra funding by VU and (DAS-5) COMMIT + NLeSC
- SURFnet (GigaPort) provides wide-area networks
Steering Committee algorithm

FOR i IN 1 .. 5 DO

Develop vision for DAS[i]
NWO/M proposal by 1 September [4 months]
Receive outcome (accept) [6 months]
Detailed spec / EU tender [4-6 months]
Selection; order system; delivery [6 months]

Research_system := DAS[i];
Education_system := DAS[i-1] (if i>1)
Throw away DAS[i-2] (if i>2)
Wait (2 or 3 years)

DONE
## Output of the algorithm

<table>
<thead>
<tr>
<th></th>
<th>DAS-1</th>
<th>DAS-2</th>
<th>DAS-3</th>
<th>DAS-4</th>
<th>DAS-5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>Pentium Pro</td>
<td>Dual Pentium3</td>
<td>Dual Opteron</td>
<td>Dual 4-core Xeon</td>
<td>Dual 8-core Xeon</td>
</tr>
<tr>
<td><strong>LAN</strong></td>
<td>Myrinet</td>
<td>Myrinet</td>
<td>Myrinet 10G</td>
<td>QDR Infini Band</td>
<td>FDR Infini Band</td>
</tr>
<tr>
<td># CPU cores</td>
<td>200</td>
<td>400</td>
<td>792</td>
<td>1600</td>
<td>3168</td>
</tr>
<tr>
<td>1-way latency MPI (µs)</td>
<td>21.7</td>
<td>11.2</td>
<td>2.7</td>
<td>1.9</td>
<td>1.2</td>
</tr>
<tr>
<td>Max. throughput (MB/s)</td>
<td>75</td>
<td>160</td>
<td>950</td>
<td>2700</td>
<td>6000</td>
</tr>
<tr>
<td>Wide-area network</td>
<td>6 Mb/s ATM</td>
<td>1 Gb/s Internet</td>
<td>10 Gb/s Light paths</td>
<td>10 Gb/s Light paths</td>
<td>10 Gb/s Light paths</td>
</tr>
</tbody>
</table>
Earlier results

- VU: programming distributed systems
  - Clusters, wide area, grid, optical, cloud, accelerators
- Delft: resource management [CCGrid’2012 keynote]
- MultimediaN: multimedia knowledge discovery
- Amsterdam: wide-area networking, clouds, energy
- Leiden: data mining, astrophysics [CCGrid’2013 keynote]
- Astron: accelerators
DAS-1 (1997-2002)
A homogeneous wide-area system

200 MHz Pentium Pro
Myrinet interconnect
BSDI → Redhat Linux
Built by Parsytec

VU (128 nodes)
Amsterdam (24 nodes)

Leiden (24 nodes)
Delft (24 nodes)

Built by Parsytec
Albatross project

- Optimize algorithms for wide-area systems
  - Exploit hierarchical structure $\Rightarrow$ locality optimizations
- Compare:
  - 1 small cluster (15 nodes)
  - 1 big cluster (60 nodes)
  - wide-area system (4×15 nodes)
Sensitivity to wide-area latency and bandwidth

- Used local ATM links + delay loops to simulate various latencies and bandwidths [HPCA’99]
Wide-area programming systems

- **Manta:**
  - High-performance Java [TOPLAS 2001]

- **MagPle (Thilo Kielmann):**
  - MPI’s collective operations optimized for hierarchical wide-area systems [PPoPP’99]

- **KOALA (TU Delft):**
  - Multi-cluster scheduler with support for co-allocation
DAS-2 (2002-2006)
a Computer Science Grid

two 1 GHz Pentium-3s
Myrinet interconnect
Redhat Enterprise Linux
Globus 3.2
PBS → Sun Grid Engine
Built by IBM

VU (72) – Amsterdam (32)
Leiden (32) – Delft (32)
Utrecht (32)

SURFnet 1 Gb/s
**Grid programming systems**

- Satin (Rob van Nieuwpoort):
  - Transparent divide-and-conquer parallelism for grids
    - Hierarchical computational model fits grids [TOPLAS 2010]
- Ibis: Java-centric grid computing [IEEE Computer 2010]
  - Efficient and transparent "Jungle computing"
- JavaGAT:
  - Middleware-independent API for grid applications [SC’07]
Grid experiments

- Do clean performance measurements on DAS
- Combine DAS with EU grids to test heterogeneity
  - Show the software “also works” on real grids
DAS-3 (2006-2010)
An optical grid

Dual AMD Opterons
2.2-2.6 GHz
Single/dual core nodes
Myrinet-10G
Scientific Linux 4
Globus, SGE
Built by ClusterVision

VU (85)

TU Delft (68)

Leiden (32)

UvA/MultimediaN (40/46)

SURFnet6

10 Gb/s
- Multiple dedicated 10G light paths between sites
- Idea: dynamically change wide-area topology
**Distributed Model Checking**

- Huge state spaces, bulk asynchronous transfers
- Can efficiently run DiVinE model checker on wide-area DAS-3, use up to 1 TB memory

[IPDPS’09]
Required wide-area bandwidth

![Graph showing WAN throughput vs. total number of cores for different core configurations.](image)
DAS-4 (2011) Testbed for Clouds, diversity, green IT

Dual quad-core Xeon E5620 Infiniband
Various accelerators
Scientific Linux
Bright Cluster Manager
Built by ClusterVision

VU (74)

ASTRON (23)

TU Delft (32)

Leiden (16)

SURFnet6

10 Gb/s

UvA/MultimediaN (16/36)
Global Climate Modeling

- Understand future local sea level changes
- Needs high-resolution simulations
- Combine two approaches:
  - Distributed computing (multiple resources)
  - GPUs
Distributed Computing

- Use Ibis to couple different simulation models
  - Land, ice, ocean, atmosphere
- Wide-area optimizations similar to Albatross project (16 years ago), like hierarchical load balancing
Enlighten Your Research Global award

10G 10G 10G

#7 STAMPEDE (USA) CARTESIUS (NLD) SUPERMUC (GER)

#10 KRAKEN (USA) EMERALD (UK)
**GPU Computing**

- Offload expensive kernels for Parallel Ocean Program (POP) from CPU to GPU
  - Many different kernels, fairly easy to port to GPUs
  - Execution time becomes virtually 0
- New bottleneck: moving data between CPU & GPU

![Diagram showing CPU, GPU, host memory, device memory, and PCI Express link.](image-url)
Problem

- Problem:
  - Which method will be most efficient for a given GPU kernel? Implementing all can be a large effort

- Solution:
  - Created a performance model that identifies the best implementation:
    - What implementation strategy for overlapping computation and communication is best for my program?

Ben van Werkhoven, Jason Maassen, Frank Seinstra & Henri Bal: Performance models for CPU-GPU data transfers, CCGrid2014
**DAS-5 (June 2015): Harnessing diversity, data-explosion**

- Same 5 sites as DAS-4
- Two new (co-funding) participants
  - Netherlands e-Science Center (NLeSC)
  - COMMIT/: a public-private research community
DAS-5

Dual 8-core Intel E5-2630v3 CPUs
FDR InfiniBand
OpenFlow switches
Various accelerators
CentOS Linux
Bright Cluster Manager
Built by ClusterVision

VU (68)

ASTRON (9)

TU Delft (48)

Leiden (24)

UvA/MultimediaN (18/31)

10 Gb/s

SURFnet7
Highlights of DAS users

- Awards
- Grants
- Top-publications
- New user-communities
Awards

- 3 CCGrid SCALE awards
  - 2008: Ibis: ``Grids as promised"
  - 2010: WebPIE: web-scale reasoning
  - 2014: BitTorrent analysis

- Video and image retrieval
  - 5 TRECVID awards, ImageCLEF, ImageNet, Pascal VOC classification, AAAI 2007 most visionary research award

- Euro-Par 2014 achievement award
Grants

- Externally funded PhD/postdoc projects using DAS:
  - DAS-3 proposal: 20
  - DAS-4 proposal: 30
  - DAS-5 proposal: 50

- Major incentive for VL-e (20 M€ funding)
  - Virtual Laboratory for e-Science (2003-2009)
- Strong participation in COMMIT/
- 100 completed PhD theses
Recent DAS papers

- Prebaked uVMs: Scalable, Instant VM Startup for IaaS Clouds (ICDCS’15)
- Cashmere: Heterogeneous Many-Core Computing (IPDPS’15)
- Glasswing: MapReduce on Accelerators (HPDC’14 / SC’14)
- Performance models for CPU-GPU data transfers (CCGrid’14)
- Auto-Tuning Dedispersion for Many-Core Accelerators (IPDPS’14)
- How Well do Graph-ProcessingPlatforms Perform? (IPDPS’14)
- Balanced resource allocations across multiple dynamic MapReduce clusters (SIGMETRICS ‘14)
- Squirrel: Virtual Machine Deployment (SC’13 + HPDC’14)
## Top papers

<table>
<thead>
<tr>
<th>Journal</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE Computer</td>
<td>5</td>
</tr>
<tr>
<td>Comm. ACM</td>
<td>2</td>
</tr>
<tr>
<td>IEEE TPDS</td>
<td>7</td>
</tr>
<tr>
<td>ACM TOPLAS</td>
<td>3</td>
</tr>
<tr>
<td>ACM TOCS</td>
<td>4</td>
</tr>
<tr>
<td>Nature</td>
<td>2</td>
</tr>
</tbody>
</table>
### SIGOPS 2000 paper

**Title:** The distributed ASCI supercomputer project  

**Publication date:** 2000/10/1  
**Journal name:** ACM SIGOPS Operating Systems Review  
**Volume:** 34  
**Issue:** 4  
**Pages:** 76-96  
**Publisher:** ACM  
**Abstract:** The Distributed ASCI Supercomputer (DAS) is a homogeneous wide-area distributed system consisting of four cluster computers at different locations. DAS has been used for research on communication software, parallel languages and programming systems, schedulers, parallel applications, and distributed applications. The paper gives a preview of the most interesting research results obtained so far in the DAS project.

**Total citations:** 130 citations

**Citations per year:**

- 2000: 3
- 2002: 16
- 2006: 5
- 2010: 2
- 2014: 1

**Scholar articles:**

- The distributed ASCI supercomputer project
  - Cited by 130 - Related articles - All 17 versions
Forthcoming paper (2015)

The Distributed ASCI Supercomputer: a Long-term Computer Science Research Infrastructure

Henri Bal  VU University, Amsterdam
Dick Epema  Delft University of Technology
Cees de Laat  University of Amsterdam
Rob van Nieuwpoort  Netherlands eScience Center
John Romein  ASTRON
Frank Seinstra  Netherlands eScience Center
Cees Snoek  University of Amsterdam
Harry Wijshoff  University of Leiden
New user-communities

- DAS is totally unattractive for applications research
  - It comes nowhere near the TOP-500
  - It cannot be used for production runs during day-time
- Still, DAS keeps attracting new communities
  - DAS-3: multimedia; DAS-4: astronomy; DAS-5: eScience
- Reason: DAS is useful as stepping stone
  - Easy and fast experiments with parallel algorithms
  - Distributed experiments
  - Multimodal networking
  - ASTRON
My background

- Cluster computing
  - Zoo (1994), Orca
- Wide-area computing
  - DAS-1 (1997), Albatross
- Grid computing
  - DAS-2 (2002), Manta, Satin
- eScience & optical grids
  - DAS-3 (2006), Ibis
- Many-core accelerators
  - DAS-4 (2010), MCL, Glasswing
  - DAS-5 (2015), Cashmere
Part II: DAS-5 - Harnessing the Diversity of Complex e-Infrastructures

- e-Infrastructures are becoming more and more distributed and diverse
  - Variety of accelerators
  - Variety of software-defined networks
- Many applications need to harness their power to address big data processing needs
- Handling the complexity and diversity of distributed infrastructures thus is a key research challenge
- We have >50 PhD/Postdoc projects in this area
Research agenda for DAS-5

- e-Infrastructure management
  - Programmable networks, resource management, GreenIT
- Harnessing Diversity
  - Programming systems for accelerators
  - Numerous eScience applications
- Interacting with big data
  - Semantic web, sensor networks, eHealth, life sciences
- Multimedia and games
  - Multimedia content analysis
- Astronomy
Agenda for Part II
Distributed computing + accelerators

- GPU programming
- MCL: A methodology for programming accelerators
- Cashmere: Heterogeneous Many-Core Computing
- Glasswing: MapReduce on accelerators
Graphics Processing Units

- GPUs and other accelerators take top-500 by storm
- Many application success stories
- But GPUs are very difficult to program and optimize

http://www.nvidia.com/object/tesla-case-studies.html
Example: convolution

- About half a Ph.D. thesis
Parallel Programming Lab course

- Lab course for MSc students (next to lectures)
- CUDA:
  - Simple image processing application on 1 node
- MPI:
  - Parallel all pairs shortest path algorithms
- CUDA: 40-52 % passed over last 2 years
- MPI: 80 % passed
Questions

- Why are accelerators so difficult to program?
- Can we develop a better methodology for programming accelerators?
- Can we exploit heterogeneous (diverse) clusters with different types of accelerators?
- Can we handle big (out-of-core) data?
Example: Convolution operations

Image I of size $I_w \times I_h$
Filter F of size $F_w \times F_h$
Thread block of size $B_w \times B_h$

Naïve CUDA kernel:
1 thread per output pixel
Does 2 arithmetic operations and 2 loads (8 bytes)

Arithmetic Intensity (AI) = 0.25
Hierarchy of concurrent threads

Grid
Memory optimizations for tiled convolution

Threads within a block cooperatively load entire area they need into a small (e.g. 96KB) shared memory

Filter (small) goes into constant memory
Tiled convolution

Using shared memory for Tiled Convolution

16x16 thread block processing an 11x7 filter

- Arithmetic Intensity:

\[
AI = \frac{2 \times F_w \times F_h \times B_w \times B_h}{(F_w - 1 + B_w) \times (F_h - 1 + B_h) \times 4}.
\]
Analysis

- If filter size increases:
  - Arithmetic Intensity increases:
    - Kernel shifts from memory-bandwidth bound to compute-bound
      \[
      AI = \frac{2 \times F_w \times F_h \times B_w \times B_h}{(F_w - 1 + B_w) \times (F_h - 1 + B_h) \times 4}.
      \]
  - Amount of shared memory needed increases → fewer thread blocks can run concurrently on each SM
Tiling

- Each thread block computes 1xN tiles in horizontal direction
  - Increases amount of work per thread
  - Saves loading overlapping borders
  - Saves redundant instructions
  - More shared memory, fewer concurrent thread blocks

No shared memory bank conflicts
Adaptive tiling

- Tiling factor is selected at runtime depending on the input data and the resource limitations of the device
  - Highest possible tiling factor that fits within the shared memory available (depending on filter size)
- Plus loop unrolling, mem optimal configuration

Why is GPU programming hard?

- Mapping algorithm to architecture is difficult, especially as the architecture is difficult:
  - Many levels of parallelism
  - Limited resources (registers, shared memory)
    - Less of everything than CPU (except parallelism), especially per thread, makes problem-partitioning difficult
  - Everything must be in balance to obtain performance

- Portability
  - Optimizations are architecture-dependent, and the architectures change frequently
Why is GPU programming hard?

- Many crucial high-impact optimizations needed:
  - Data reuse
    - Use shared memory efficiently
    - Limited by #registers per thread, shared memory per thread block
  - Memory access patterns
    - Shared memory bank conflicts, global memory coalescing
  - Instruction stream optimization
    - Control flow divergence, loop unrolling
  - Moving data to/from the GPU
Why is GPU programming hard?

- Bottom line: tension between
  - control over hardware to achieve performance
  - higher abstraction level to ease programming
- Programmers need understandable performance
- Old problem in Computer Science, but now in extreme form
Programming methodology: stepwise refinement for performance

- Methodology:
  - Programmers can work on *multiple levels of abstraction*
  - Integrate *hardware descriptions* into programming model
  - *Performance feedback* from compiler, based on hardware description and kernel
  - Cooperation between compiler and programmer

MCL: Many-Core Levels

- MCL program is an algorithm mapped to hardware

- Start at a suitable abstraction level
  - E.g. idealized accelerator, NVIDIA Kepler GPU, Xeon Phi

- MCL compiler guides programmer which optimizations to apply on given abstraction level or to move to deeper levels
MCL ecosystem

Eclipse plugin

- MCL program
- Hardware description

Compiler

- Translate between abstraction-levels
- Generate feedback
- Generate code

- MCL program
- Performance feedback
- OpenCL and C++ code

Powered by Rascal
parallelism hierarchy {
    memory_space main {
    }
    par_group threads {
        nr_units = unlimited;
        par_unit thread {
        }
    }
}

device perfect {
    mem;
    ic;
    cores;
}

memory mem {
    space(main);
    capacity = unlimited B;
}

interconnect ic {
    connects(mem, cores.core[*]);
    latency = 1 cycle;
    bandwidth = unlimited bits/s;
}

execution_group cores {
    nr_units = unlimited;
    execution_unit core {
        slots(thread, 1);
        instructions ops {
            instruction(\(+\), 1 cycle);
            instruction(\(/\), 1 cycle);
            instruction(\(-\), 1 cycle);
            instruction(\(*\), 1 cycle);
        }
    }
}

import perfect;

perfect void convolve(int h, int w, int fh, int fw,
    main float [h, w] output,
    main float [h + fh/2*2, w + fw/2*2] input,
    main float [fh, fw] filter) {

    foreach (int i in h threads ) {
        foreach(int j in w threads ) {
            float sum = 0.0;
            for (int y = 0; y < fh; y++) {
                for (int x = 0; x < fw; x++) {
                    sum += filter[y, x] * input[i + y, j + x];
                }
            }
            output[i, j] = sum / (fh * fw);
        }
    }
}
Compiler feedback

129 GFLOPS

positive data reuse ratio in input, use shared memory:
214 GFLOPS

Try to maximize #blocks per SMP, now using 1 of 8 blocks: 240 GFLOPS

sum[k,l] may benefit from cache:
worst case is 2 cache line fetches: 302 GFLOPS
Performance
(GTX480, 9×9 filters)

Performance of different 2D Convolution kernels on GTX480

380 GFLOPS
MCL: 302 GFLOPS
Compiler +
Performance evaluation

<table>
<thead>
<tr>
<th>Application</th>
<th>Naive</th>
<th>MCL</th>
<th>Other implementations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NVIDIA GTX480 GPU</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>matmul</td>
<td>89.0 GFLOPS</td>
<td>564 GFLOPS</td>
<td>892 GFLOPS</td>
</tr>
<tr>
<td>convolution</td>
<td>129 GFLOPS</td>
<td>302 GFLOPS</td>
<td>380 GFLOPS</td>
</tr>
<tr>
<td>histogram</td>
<td>6.94 GB/s</td>
<td>80.0 GB/s</td>
<td>80 GB/s</td>
</tr>
<tr>
<td>gesummv*</td>
<td>1.25 GFLOPS</td>
<td>51.9 GFLOPS</td>
<td>2.98 GFLOPS</td>
</tr>
<tr>
<td>blackScholes</td>
<td>297 GFLOPS</td>
<td>432 GFLOPS</td>
<td>237 GFLOPS</td>
</tr>
<tr>
<td>sparse matmul</td>
<td>1.01 GFLOPS</td>
<td>3.88 GFLOPS</td>
<td>8.9 GFLOPS</td>
</tr>
<tr>
<td><strong>Intel Xeon Phi (5110P)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>matmul</td>
<td>39.9 GFLOPS</td>
<td>488 GFLOPS</td>
<td>695 GFLOPS</td>
</tr>
<tr>
<td>convolution</td>
<td>66.4 GFLOPS</td>
<td>171 GFLOPS</td>
<td>n/a</td>
</tr>
<tr>
<td>histogram**</td>
<td>0.331 GB/s</td>
<td>11.9 GB/s</td>
<td>0.3 GB/s</td>
</tr>
<tr>
<td>gesummv</td>
<td>0.84 GFLOPS</td>
<td>55.4 GFLOPS</td>
<td>n/a</td>
</tr>
<tr>
<td>blackScholes</td>
<td>66 GFLOPS</td>
<td>115 GFLOPS</td>
<td>n/a</td>
</tr>
<tr>
<td>sparse matmul**</td>
<td>3.93 GFLOPS</td>
<td>5.28 GFLOPS</td>
<td>13 GFLOPS</td>
</tr>
</tbody>
</table>

Compared to known, fully optimized versions
(* measured on a C2050, ** using a different input).
Lessons learned from MCL

- The stepwise-refinement for performance methodology supports a structured approach
  - Gradually expose programmers to more hardware details
- It enables to develop optimized many-core kernels
- Key ideas: stepwise refinement + multiple abstraction levels + compiler feedback
Heterogeneous many-core clusters

- New GPUs become available frequently, but older-generation GPUs often still are fast enough
  - Clusters become heterogeneous and contain different types of accelerators
- VU DAS-4 cluster
  - NVIDIA GTX480 GPUs (22)
  - NVIDIA K20 GPUs (8)
  - Intel Xeon Phi (2)
  - NVIDIA C2050 (2), Titan, GTX680 GPU
  - AMD HD7970 GPU
**Heterogeneity in TOP-500**
*(list of Nov. 2014)*

<table>
<thead>
<tr>
<th>name</th>
<th>ranking</th>
<th>configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quartetto</td>
<td>49</td>
<td>K20, K20X, Xeon Phi 5110P</td>
</tr>
<tr>
<td>Lomonosov</td>
<td>58</td>
<td>2070, PowerXCell 8i</td>
</tr>
<tr>
<td>HYDRA</td>
<td>77</td>
<td>K20X, Xeon Phi</td>
</tr>
<tr>
<td>SuperMIC</td>
<td>88</td>
<td>Xeon Phi 7110P, K20X</td>
</tr>
<tr>
<td>Palmetto2</td>
<td>89</td>
<td>K20m, M2075, M2070</td>
</tr>
<tr>
<td>Armstrong</td>
<td>103</td>
<td>Xeon Phi 5120D, K40</td>
</tr>
<tr>
<td>Loewe-CSC</td>
<td>179</td>
<td>HD5870, FirePro S10000</td>
</tr>
<tr>
<td>Inspur</td>
<td>310</td>
<td>K20m, Xeon Phi 5110P</td>
</tr>
<tr>
<td>Tsubame 2.5</td>
<td>392</td>
<td>K20X, S1070, S2070</td>
</tr>
<tr>
<td>El Gato</td>
<td>465</td>
<td>K20, K20X, Xeon Phi 5110P</td>
</tr>
</tbody>
</table>
Cashmere

- Programming system for heterogeneous many-core clusters
- Integrates two systems
  - Satin: divide-and-conquer Grid programming system
  - MCL: allows kernels to be written and optimized for each type of hardware
- Cashmere does integration, application logic, mapping, and load balancing for multiple GPUs/node

Pieter Hijma, Rob van Nieuwoort, Ceriel Jacobs, Henri Bal, 29th IEEE Int. Parallel & Distributed Processing Symposium (IPDPS 2015)
Satin

- Model
  - Divide-and-conquer programming model, inspired by Cilk
  - Targets hierarchical distributed systems (e.g., grids)
- Advantages
  - Load-balancing (cluster-aware random work-stealing)
  - Latency hiding, especially on wide-area links
  - Good scalability on wide-area systems
Cashmere programming model

- Additional levels of parallelism
  - Parallelism on many-core device
  - Multiple many-core devices per compute node
  - Overlap in communication and computation

- Goal:
  - Use these additional levels of parallelism without disruptive changes to the Satin programming model
Cashmere skeleton

```java
int spawnable f(a) {
    if (small_enough_for_leaf(a)) {
        return do_leaf_computation(a)
    }
    else if (small_enough_for_many_core(a)) {
        Cashmere.enableManyCore()
    }

    r1 = f(make_smaller(a)) //asynchronous
    r2 = f(make_smaller(a)) //asynchronous
    sync

    return combine(r1, r2)
}
```
Evaluation

- Methodology
  - Measure kernel performance
  - Measure scalability on homogeneous cluster
  - Compare efficiency homogeneous/heterogenous

- Application classes

<table>
<thead>
<tr>
<th>Application</th>
<th>Type</th>
<th>Computation</th>
<th>communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raytracer</td>
<td>Irregular</td>
<td>heavy</td>
<td>light</td>
</tr>
<tr>
<td>Matmul</td>
<td>Regular</td>
<td>heavy</td>
<td>heavy</td>
</tr>
<tr>
<td>k-means</td>
<td>Iterative</td>
<td>moderate</td>
<td>light</td>
</tr>
<tr>
<td>n-body</td>
<td>Iterative</td>
<td>heavy</td>
<td>moderate</td>
</tr>
</tbody>
</table>
Kernel performance (GFLOP/s)
K-Means on a homogeneous GTX480 cluster

scalability

absolute performance
## Heterogeneous performance

<table>
<thead>
<tr>
<th>application</th>
<th>performance (GFLOPS)</th>
<th>configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>raytracer</td>
<td>1883</td>
<td>10 gtx480, 2 c2050, 1 gtx680, 1 titan, 1 hd7970</td>
</tr>
<tr>
<td>matmul</td>
<td>3927</td>
<td>10 gtx480, 2 c2050, 1 gtx680, 1 titan, 1 hd7970</td>
</tr>
<tr>
<td>k-means</td>
<td>10644</td>
<td>10 gtx480, 2 c2050, 1 gtx680, 1 titan, 1 hd7970, 7 k20, 1 xeon phi</td>
</tr>
<tr>
<td>n-body</td>
<td>7002</td>
<td>10 gtx480, 2 c2050, 1 gtx680, 1 titan, 1 hd7970</td>
</tr>
</tbody>
</table>

**Homogeneous:** efficiency on 16 GTX480

**Heterogeneous:** efficiency over total combined hardware
Lessons learned on Cashmere

- Seamless integration of many-cores in Satin
- MCL enabled writing/optimizing variety of kernels
- High performance and automatic load balancing even when the many-core devices differ widely
- Efficiency >90% in 3 out of 4 applications in heterogeneous executions
**Future work on MCL/Cashmere**

- Improve MCL compiler analysis
- Include performance prediction models
- Managing different versions at different levels
- Real-world applications
  - Digital forensics
  - Climate modeling
  - Natural language processing
  - High-energy physics (LHC)
- Extend Cashmere model beyond divide & conquer
- Extend implementation to wide-area systems
- Extend to out-of-core (big data) applications
Other approaches that deal with performance vs abstraction

- Domain specific languages
- Patterns, skeletons, frameworks
- Berkeley Dwarfs

List of Dwarfs

1. Dense Linear Algebra
2. Sparse Linear Algebra
3. Spectral Methods
4. N-Body Methods
5. Structured Grids
6. Unstructured Grids
7. MapReduce
8. Combinational Logic
9. Graph Traversal
10. Dynamic Programming
11. Backtrack and Branch-and-Bound
12. Graphical Models
13. Finite State Machines
MapReduce

- Big Data revolution
- Designed for cheap commodity hardware
- Scales horizontally
- Coarse-grained parallelism
- MapReduce on modern hardware?
Glasswing: Rethinking MapReduce

- Use accelerators (OpenCL) as mainstream feature
- Massive out-of-core data sets
- Scale vertically & horizontally
- Maintain MapReduce abstraction

Ismail El Helw, Rutger Hofman, Henri Bal [HPDC’2014, SC’2014]
## Related work

<table>
<thead>
<tr>
<th></th>
<th>Out of Core</th>
<th>Compute Device</th>
<th>Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phoenix</td>
<td>×</td>
<td>CPU-only</td>
<td>×</td>
</tr>
<tr>
<td>Tiled-MapReduce</td>
<td>×</td>
<td>NUMA CPU</td>
<td>×</td>
</tr>
<tr>
<td>Mars</td>
<td>×</td>
<td>GPU-only</td>
<td>×</td>
</tr>
<tr>
<td>Ji et al.</td>
<td>×</td>
<td>GPU-only</td>
<td>×</td>
</tr>
<tr>
<td>MapCG</td>
<td>×</td>
<td>CPU/GPU</td>
<td>×</td>
</tr>
<tr>
<td>Chen et al.</td>
<td>×</td>
<td>GPU-only</td>
<td>×</td>
</tr>
<tr>
<td>GPMR</td>
<td>×</td>
<td>GPU-only</td>
<td></td>
</tr>
<tr>
<td>Chen et al.</td>
<td>×</td>
<td>AMD Fusion</td>
<td>×</td>
</tr>
<tr>
<td>Merge</td>
<td>×</td>
<td>Any</td>
<td>×</td>
</tr>
<tr>
<td>HadoopCL</td>
<td>✓</td>
<td>APARAPI</td>
<td>✓</td>
</tr>
<tr>
<td>Glasswing</td>
<td>✓</td>
<td>OpenCL enabled</td>
<td>✓</td>
</tr>
</tbody>
</table>
Glasswing Pipeline

- Overlaps computation, communication & disk access
- Supports multiple buffering levels

<table>
<thead>
<tr>
<th>Interaction with compute device</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Map Pipeline</strong></td>
</tr>
<tr>
<td>Read input split</td>
</tr>
<tr>
<td><strong>Reduce Pipeline</strong></td>
</tr>
<tr>
<td>Read intermediate data</td>
</tr>
</tbody>
</table>
GPU optimizations

- Glasswing framework does:
  - Memory management
  - Some shared memory optimizations
  - Data movement, data staging

- Programmer:
  - Focusses on the map and reduce kernels (using OpenCL)
  - Can do kernel optimizations if needed
    - Coalescing, memory banks, etc.
Glasswing vs. Hadoop
64-node CPU Infiniband cluster

![Bar chart comparing Glasswing and Hadoop for different tasks such as Word Count, Page View Count, TeraSort, Matrix Multiply, and K-Means, with Glasswing generally performing faster except for TeraSort where Hadoop is significantly faster.](chart.png)
Glasswing vs. Hadoop
16-Node GTX480 GPU Cluster

![Graph showing comparison between Glasswing and Hadoop in Matrix Multiply and K-Means tasks.](image)
Compute Device Comparison

K-Means

<table>
<thead>
<tr>
<th>Device</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xeon CPU</td>
<td>1400</td>
</tr>
<tr>
<td>Xeon Phi</td>
<td>1200</td>
</tr>
<tr>
<td>GTX480</td>
<td>400</td>
</tr>
<tr>
<td>GTX680</td>
<td>100</td>
</tr>
<tr>
<td>K20</td>
<td>10</td>
</tr>
</tbody>
</table>

Matrix Multiply

<table>
<thead>
<tr>
<th>Device</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xeon CPU</td>
<td>600</td>
</tr>
<tr>
<td>Xeon Phi</td>
<td>300</td>
</tr>
<tr>
<td>GTX480</td>
<td>100</td>
</tr>
<tr>
<td>GTX680</td>
<td>50</td>
</tr>
<tr>
<td>K20</td>
<td>50</td>
</tr>
</tbody>
</table>
Lessons learned on Glasswing

- MapReduce framework combining coarse-grained and fine-grained parallelism
- Scales vertically & horizontally
- Handles out-of-core data, sticks with MapReduce model
- Overlaps kernel executions with memory transfers, network communication and disk access
- Outperforms Hadoop by 1.2 – 4x on CPUs and 20 – 30x on GPUs
Discussion on programming accelerators

- Accelerator programming and optimization currently is too time-consuming for many applications
  - GPU programs need many complex optimizations to obtain high performance
- Many (eScience) applications do need performance of modern accelerators
  - Next in line: SKA, high-energy physics, forensics, water management, ecology, natural language processing, ...
- How to deal with the tension between abstraction-level and control?
  New programming methodologies that allow a choice
Discussion on Infrastructure

- Computer Science needs its own infrastructure for interactive experiments
  - DAS was instrumental for numerous projects
- Being organized helps
  - ASCI is a (distributed) community
- Having a vision for each generation helps
  - Updated every 4-5 years, in line with research agendas
- Other issues:
  - Expiration date?
  - Size?
Expiration date

- Need to stick with same hardware for 4-5 years
  - Also cannot afford expensive high-end processors
- Reviewers sometimes complain that the current system is out-of-date (after > 3 years)
  - Especially in early years (clock speeds increased fast)
- DAS-4/DAS-5: accelerators added during the project
Does size matter?

- Reviewers seldom reject our papers for small size
  - “This paper appears in-line with experiment sizes in related SC research work, if not up to scale with current large operational supercomputers.”
- We sometimes do larger-scale runs in clouds or on production supercomputers
Interacting with applications

- Used DAS as *stepping stone* for applications
  - Small experiments, no productions runs
- Applications really helped the CS research
  - DAS-3: multimedia → Ibis applications, awards
  - DAS-4: astronomy → many new GPU projects
  - DAS-5: eScience Center → EYR-G award, GPU work
- Many plans for new collaborations
  - Data science
  - Business analytics
Conclusions

- DAS has had a major impact on Dutch Computer Science for almost two decades
- It aims at coherence + modest system size
- Highly effective and small investment
  - E.g. It avoids fragmented systems management
- Central organization and coherence are key
Acknowledgements

**DAS Steering Group:**
Dick Epema  
Cees de Laat  
Cees Snoek  
Frank Seinstra  
John Romein  
Harry Wijshoff

**System management:**
Kees Verstoep et al.

**Hundreds of users**

**Support:**
TUD/GIS  
Stratix  
ASCI office

**DAS grandfathers:**
Andy Tanenbaum  
Bob Hertzberger  
Henk Sips

**More information:**
http://www.cs.vu.nl/das5/

**Funding:**
Different methods for CPU-GPU communication

- Memory copies (explicit)
  - No overlap with GPU computation
- Device-mapped host memory (implicit)
  - Allows fine-grained overlap between computation and communication in either direction
- CUDA Streams or OpenCL command-queues
  - Allows overlap between computation and communication in different streams
- Any combination of the above
Example result

Measured and estimated performance on GTX 680 (PCIe 2.0)

- Explicit
- Implicit
- Streams
- Hybrid

Measured  |  Model
---|---
state-observed  |  state-estimated  |  buoydiff-observed  |  buoydiff-estimated

Time (ms)

{Graph showing performance comparison with measured and estimated data for different models and states using bar charts.}
Different GPUs (state kernel)
Different GPUs (buoydiff)
**Performance models for CPU-GPU data transfers**

1) GPU properties
   - Number of copy engines: No

2) Kernel properties
   - Host to Device transfers (bytes): 53870912
   - Kernel Execution Time (ms): 20
   - Average # loads of input elements per thread: 1

2b) Streams properties
   - Largest per stream HtoD transfer (bytes): 
   - Largest per stream DtoH transfer (bytes): 
   - Number of streams: 

3) Interconnect parameters
   - Load preset: GTX Titan
   - PCIe version: 3.0
   - Host to Device bandwidth (Gidh): 8.318E-08 ms/byte
   - Device to Host bandwidth (Gdhh): 7.925E-08 ms/byte
   - Device to Host bandwidth (Gdgh): 0.002693 ms
   - Hybrid bandwidth (mixed hograd/mapped-mem): 1.104E-07 ms

[For more information about the performance models and parameters please see:]

**Estimated kernel performance**

**Computation / Communication balance**

**Theoretical bounds**

For more information about the performance models for CPU-GPU data transfers, see:

B. van Werkhoven, J. Maassen, F.J. Semstra, and H.E. Bal
Lessons learned

- Everything must be in balance to obtain high performance
  - Subtle interactions between resource limits
- Runtime decision system (adaptive tiling), in combination with standard optimizations
  - Loop unrolling, memory bank conflicts
Why is GPU programming hard?

- Portability
  - Optimizations are architecture-dependent, and the architectures change frequently
  - Optimizations are often input dependent
- Finding the right parameters settings is difficult
- Need better performance models
  - Like Roofline and our I/O model
Performance K-Means

![Graph showing performance comparison between Hadoop, Glasswing CPU, Glasswing GPU, and GPMR compute. The graph plots time (seconds) against the number of nodes, with each technology represented by a different line.](image-url)