Computing in the Mist: Writing Applications for Unknown Machines

Thilo Kielmann
VU University, Amsterdam
kielmann@cs.vu.nl
• How many cores does your computer have?
• Where is your data?
• What has happened to job number 2342?
ITRS Roadmap 2005 and 2007

Clock Rate (GHz)

2005 Roadmap

2007 Roadmap

Intel single core

Intel multicore

[D. Patterson, USENIX 2008 keynote]
A Fundamental Technology Change

- CPU's will get faster only marginally:
  - limits of integration density
  - energy consumption (proportional to clock rate)
- If you want a faster computer, you need to use multiple CPU's:
  - In the past, the clock rate doubled every 18 months
  - In the future, the number of cores will double every 18-24 months
- All programs must be parallel to use this new hardware!
Intel's 80-Core Teraflop Chip Prototype
IBM's Cell: Heterogeneous Ensemble

Source: M. Gschwind et al., Hot Chips-17, August 2005
Example: Roadrunner

- Leading the TOP500 by using Cell processors
Example: Nvidia's GeForce8800
- 8 x 16 blocks of stream processors
- separate thread schedulers
- crossbar-like access to graphics memory
GPU vs. CPU

- **Nvidia GeForce 8800**
  - clock speed 1.35 GHz
  - 681 million transistors

- **Intel Pentium 4**
  - clock speed 2.4 GHz
  - 55 million transistors

[Schellmann et.al., Euro-PAR 2008]:

2 Nvidia GeForce 8800GTX as fast as 16 Intel Xeon 3.2GHz
General Purpose GPU Programming

Nvidia's CUDA

**Computing** $y \leftarrow ax + y$ **with a Serial Loop**

```c
void saxpy_serial(int n, float alpha, float *x, float *y)
{
    for(int i = 0; i<n; ++i)
        y[i] = alpha*x[i] + y[i];
}

// Invoke serial SAXPY kernel
saxpy_serial(n, 2.0, x, y);
```

**Computing** $y \leftarrow ax + y$ **in parallel using CUDA**

```c
__global__
void saxpy_parallel(int n, float alpha, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;

    if( i<n ) y[i] = alpha*x[i] + y[i];
}

// Invoke parallel SAXPY kernel (256 threads per block)
int nblocks = (n + 255) / 256;
saxpy_parallel<<<nbblocks, 256>>>(n, 2.0, x, y);
```
Clusters: Beyond Single Computers

DAS3, VU Amsterdam:
Programming for Clusters

- Distributed memory, high-speed networks
- commonly: message passing (MPI), C and Fortran
- academic: Java remote method invocation (Ibis)

- deployment via shared file system and batch queue scheduling
Clouds: Data Centers + Virtualization

(OK, a bit too simplified...)

VU university amsterdam
Programming Clouds: Hadoop

- implements the map-reduce paradigm
- allows processing of large data sets
- user defined map and reduce functions
- Hadoop distributed file system (HDFS) built on top of Amazon S3
- Got popular due to fault-tolerant, file-based implementation
A grid:

• Coordinates resources that are not subject to centralized control...

• ...using standard, open, general-purpose protocols and interfaces...

• ...to deliver non-trivial qualities of service.
Grid Programming:
Globus GASS copies a file...

```c
int copy_file (char const* source, char const* target) {
    globus_url_t source_url;
    globus_io_handle_t dest_io_handle;
    globus_ftp_client_operationattr_t source_ftp_attr;
    globus_result_t result;
    globus_gass_transfer_requestattr_t source_gass_attr;
    globus_gass_copy_attr_t source_gass_copy_attr;
    globus_gass_copy_handleattr_t gass_copy_handleattr;
    globus_gass_copy_handleattr_t ftp_handleattr;
    globus_io_attr_t io_attr;
    int output_file;

    if ( globus_url_parse (source_URL, &source_url) != GLOBUS_SUCCESS ) {
        printf ("can not parse source_URL \\"%s\\n", source_URL);
        return (-1);
    }

    if ( source_url.scheme_type != GLOBUS_URL_SCHEME_GSIFTP &&
         source_url.scheme_type != GLOBUS_URL_SCHEME_FTP &&
         source_url.scheme_type != GLOBUS_URL_SCHEME_HTTP &&
         source_url.scheme_type != GLOBUS_URL_SCHEME_HTTPS ) {
        printf ("can not copy from %s - wrong prot\\n", source_URL);
        return (-1);
    }

    globus_gass_copy_handleattr_init (&gass_copy_handleattr);
    globus_gass_copy_attr_init (&source_gass_copy_attr);
    globus_ftp_client_handleattr_init (&ftp_handleattr);
    globus_io_fileattr_init (&io_attr);
    globus_gass_copy_attr_set_io (&source_gass_copy_attr, &io_attr);
    globus_gass_copy_handleattr_set_ftp_attr (&gass_copy_handleattr, &ftp_handleattr);
    globus_gass_copy_handle_init (&gass_copy_handle, &gass_copy_handleattr);

    if ( source_url.scheme_type == GLOBUS_URL_SCHEME_GSIFTP ||
          source_url.scheme_type == GLOBUS_URL_SCHEME_GSIFTP ) {
        globus_ftp_client_operationattr_init (&source_ftp_attr);
        globus_gass_copy_attr_set_ftp (&source_gass_copy_attr, &source_ftp_attr);
    }
    else { // GSIFTP
        globus_gass_transfer_requestattr_init (&source_gass_attr, source_url.scheme);
        globus_gass_copy_attr_set_gass(&source_gass_copy_attr, &source_gass_attr);
    }

    output_file = globus_libc_open ((char*) target, O_WRONLY | O_TRUNC | O_CREAT,
                                     S_IRUSR  | S_IWUSR | S_IRGRP |
                                     S_IWGRP);
    if ( output_file == -1 ) {
        printf ("could not open the file \\"%s\\n", target);
        return (-1);
    }

    /* convert stdout to be a globus_io_handle */
    if ( globus_io_file_posix_convert (output_file, 0, &dest_io_handle)
         != GLOBUS_SUCCESS) {
        printf ("Error converting the file handle\\n");
        return (-1);
    }

    result = globus_gass_copy_register_url_to_handle ( &gass_copy_handle, (char*)source_URL,
                                                      &source_gass_copy_attr, &dest_io_handle,
                                                      my_callback, NULL);
    if ( result != GLOBUS_SUCCESS ) {
        printf ("error: %s\\n", globus_object_printable_to_string (globus_error_get (result)));
        return (-1);
    }

    globus_url_destroy (&source_url);
    return (0);
}
```
The Grid Application Toolkit (JavaGAT)

Features:

JavaGAT -- A Kindler Gentler Grid Interface
by Rob van Nieuwpoort and Thilo Kielmann
Vrije Universiteit, Amsterdam

[SC'07]
The Grid Application Toolkit (JavaGAT)

Grid Application

High-level API
- GAT Object Factory
- Monitoring
- Steering
- Grid I/O
- Resource Brokering
- Information system

JavaGAT Engine

Capability Provider Interface (CPI)
- Monitoring
- Steering
- Grid I/O
- Resource Brokering
- Information system

JavaGAT Adaptors
- Local
- SSH
- FTP
The Simple API for Grid Applications (SAGA): “The GAT Standardized”

- The need for a standard programming interface
  - Projects keep reinventing the wheel again, yet again, and again
  - MPI as a useful analogy of community standard
  - OGF as the natural choice; established the SAGA-RG

- Community process
  - Design and requirements derived from 23 use cases
  - SAGA Design Team (OGF, Berkeley, VU, LSU, NEC)
Ibis: Grid Programming and Deployment Simplified

**Grid Applications**
MEG Analysis, Multimedia Content Analysis, Satisfiability Solver, Automatic Grammar Learning, N-Body Simulation, etc.

**Programming Models**
Satin, MPJ, RMI, GMI

**Deployment and Management**
IbisDeploy, Adaptive Satin, Barnes GUI, etc.

**IPL**
Communication, Membership, Fault Tolerance, etc.

**JavaGAT**
Job Submission, Monitoring, File Transfer, etc.

**SmartSockets**
Robust Communication

**Traditional Communication Libraries**

**Traditional Grid Middleware**

**Zorilla**
Peer to Peer Grid Middleware
Satin: Divide-and-Conquer for the Grid

Effective paradigm for Grid applications (hierarchical)

Satin: Grid-aware load balancing (work stealing)

Also support for

Fault tolerance
Malleability
Migration
Satin Example: Fibonacci

public interface FibInter extends ibis.satin.Spawnable {
    public int fib (int n);
}

class Fib extends ibis.satin.SatinObject implements FibInter {
    public int fib (int n) {
        if (n < 2) return n;
        int x = fib(n-1); /*spawned */
        int y = fib(n-2); /*spawned, too */
        sync();
        return x + y;
    }
}

(use byte code rewriting to generate parallel code)
More Mist Programming Paradigms: Master-Worker Parallelism

- “Embarrassingly parallel” problems
- minimal communication
- no dependence on numbers/types of computers

- Popular e.g. by Seti-at-home, BOINC, etc.
- only(?) applicable to very simple problems
Challenges of “Misty” Platforms

- **Scalability**
  - applications have to run on widely different numbers of CPU's
  - if your program can not use twice the number of CPU's, you won't be able to utilize next year's computer

- **Heterogeneity**
  - applications will have to run on many core and multi core, and special-purpose CPU's (like Cell and GPU's)
  - think of clusters of multi core, clusters of Cell's, clusters of clusters of...
Challenges (2)

- **Performance portability**
  - applications must run *efficiently* on different types of machines (one of the hard problems of parallel computing)
  - I mean, both on Tsubame with GPGPU's *and* on Roadrunner with Cell's...

- **Malleability**
  - applications must be able to run with changing numbers of processors, at run time
  - adapt to changing environments

- **Fault tolerance**
  - simple statistics: with a large number of parts involved, failure probability raises towards 1
Lots of heroic efforts squeezing out performance:
- CUDA
- Cell
- Astron writing Assembler for the Blue Gene/L ...

We are back to the (Transputer) times where codes were written for specific parallel machines
- Not what we want (except for researching machines)

The opposite on clouds and grids: (lazy guys)
- Map/Reduce and Hadoop abstract from machine
- but add fault tolerance and malleability
We need a lower layer of efficiency primitives that handle certain platforms the respectively “best” way
- this is for the CCI's, CCR's, IPL's, Pthreads, ...
- users should not see this, only tool writers or compilers

A higher “coordination” layer has to describe available concurrency, in a declarative manner (?)

This means, we should reconsider the works from the 80s and 90s and see why they failed and what we could use today
- map-reduce as the perfect example for an old idea, re-animated
- There is hope for our/my/your Ph.D. work, after all ;-)
Summary / Conclusions

- The future is parallel
- Parallel programming is hard
- This is a big chance for Computer Science to get it right, finally...

Approach:
- The solution will be a combination of declarative parallelism, combined with *MUCH* systems work on getting the plumbing right
- We might have to step back from getting the last bit of performance in favour of a more sustainable approach